Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Word and Paragraph Embeddings for Expressive Speech Synthesis

A Degree Thesis Submitted to the Faculty of the Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Submitted by

German Gomez Bajo

In partial fullfillment of the requirements for the degree in Audiovisual Systems Engineering

Advisors: Antonio Bonafonte and Santiago Pascual de la Puente

Universitat Politècnica de Catalunya (UPC)
2016 - 2017

Abstract

Speech synthesis is the task of generating speech using computers. Due to the limitations of classical techniques, these systems are normally not suitable for applications that would benefit from expressiveness in the speech, such as audiobook reading.

In this project, we attempt to develop a text-to-speech speech synthesizer that is capable of reacting to the semantic content of the input text to produce expressive speech. The system is based on the Socrates text-to-speech framework developed in the VEU research lab at UPC and the Keras deep learning library.

The first part of the project was to develop a baseline system based on RNN-LSTM that doesn't take into account the semantic content of the text. Once we had this baseline system working, the additions for the expressive speech were developed. Throughout the development, the Blizzard Challenge 2013 text-to-speech corpus, which contains audiobooks, was used to train the systems. This corpus was chosen because of its richness in expressive speech.

To develop the expressive speech, we used text classification to predict the meaning of a given sentence, and used this information to improve the baseline system. This prediction is trained using a Stanford dataset with movie reviews. Because the type of text on this dataset is different from the Blizzard one, there is a domain adaptation that is performed to transfer information from the Blizzard corpus into the Stanford one.

Three experiments have been carried out to find the set of expressive features that give the best objective and subjective results by means of an evaluation done by nine volunteers. According to the objective metrics, the baseline system is the one that performs best with the Blizzard corpus, but the subjective evaluation shows some preference for the modified systems.

Resumen

La sintesis de voz consiste en utilizar ordenadores para generar voz humana. Debido a las limitaciones de las técnicas clásicas, estos sitemas normalmente no son adecuados para aplicaciones que requieren voz expresiva como en la lectura automática de audiolibros.

En este proyecto, tratamos de desarrollar un sintetizador de voz capaz de reaccionar al contenido semántico del texto para producir voz expresiva. El sistema está basado en el framework de síntesis de voz Socrates, desarrollado en el grupo VEU de la UPC, y en la librería de deep learning Keras.

La primera parte del proyecto consiste en desarrollar un sistema base que no tiene en cuenta el contenido semántico, basado en redes neuronales recurrentes RNN-LSTM. Una vez finalizada esta parte, se continuó con el desarrollo de la síntesis expresiva. Durante el proyecto hemos usado la base de datos Blizzard Challenge 2013, la cual contiene una serie de audiolibros. Elegimos esta base de datos en particular por ser muy rica en expresividad de la voz.

Para desarrollar la síntesis de voz expresiva, usamos procesado del lenguaje natural (PLN) para predecir el significado de las frases con un clasificador. Usamos esta informacion para mejorar el sistema anterior. Esta parte se hizo usando una base de datos llamada Stanford sentiment treebank, la cual contiene una serie de reseñas de cine. Debido a que esta base de datos contiene texto de distinta naturaleza del de Blizzard, hacemos una adaptación del dominio del clasificador.

Se han realizado tres experimento para su posterior evaluación objetiva y subjetiva, la segunda preguntando a nueve personas voluntarias. Segun los resultados objetivos, el sistema base es el que tiene mejores resultados, pero segun los subjetivos, hay una preferencia por los experimentos expresivos.

Resum

La síntesi de veu consisteix en fer servir ordinadors per generar veu humana. Degut a les limitacions de les tècniques clàssiques, aquests sistemes normalment no són adequats per aplicacions que requireixen veu expressiva com és el cas de la lectura de audiollibres automàtica.

En aquest projecte, desenvolupem un sintetitzador de veu capaç de reaccionar al contingut semàntic del text per produir veu expressiva. El sistema està basat en el framework de síntesi de veu Socrates, desenvolupat al grup de recerca VEU de la UPC, i en la llibreria de deep learning Keras.

La primera part del projecte consisteix en desenvolupar un sistema base que no tingui en compte el contingut semàntic del text, basat en xarxes neuronals recurrents RNN-LSTM. Finalitzada aquesta part, es va continuar amb el desenvolupament de la síntesi de veu expressiva. Durant aquest projecte hem fet servir la base de dades Blizzard Challenge 2013 per fer la síntesi, que conté audiollibres. Vam escollir aquesta base de dades en particular per la seva riquesa en veu expressiva.

Per desenvolupar la síntesi de veu expressiva, vam fer servir processat del llenguatge natural (PLN) per predir el significat de les frases amb un classificador. Vam fer servir aquesta informació per millorar el sistema anterior. En aquesta part, vam fer servir la base de dades Stanford sentiment treebank, que conté una serie de ressenyes de cinema. Degut a que el text d'aquesta base de dades es bastant diferent del Blizzard, semànticamente, es va fer una adaptació del classificador per poder fer-lo servir en la síntesi.

S'han realitzat i evaluat tres experiments de síntesi i s'han valorar objectiva y subjectivamnt, aquesta segona preguntant a nou voluntaris. Els resultats objectius mostren que el sistema base és objectivament millor però els subjectius mostren una preferència pels experiments expressius.

Acknowledgements

First of all, I would like to thank my two project advisors for being so patient with me and enabling me to do this project. Without their support and ideas and all the resources that they provided me with, this project would've been impossible to finish.

Secondly, thanks to the VEU research lab and the people in charge of the computing resources of the D5 building for giving me a place to work in their servers. And last but definitely not least, I would also like to thank everyone who so kindly participated in the subjective evaluation test.

Revision history and approval record

Revision	Date	Purpose
0	19/06/2017	Document creation
1	03/07/2017	Document revision
2	05/07/2017	Document approval

DOCUMENT DISTRIBUTION LIST

Name	e-mail
German Gomez	german.gomez.bajo@alu-etsetb.upc.edu
Santiago Pascual de la Puente	santiago.pascual@tsc.upc.edu
Antonio Bonafonte	antonio.bonafonte@upc.edu

Written by:		Reviewed and approved by:		Reviewed and approved by:	
Date	19/06/2017	**Date**	05/07/2017	**Date**	05/07/2017
Name	German Gomez	**Name**	Antonio Bonafonte	**Name**	Santiago Pascual de la Puente
Position	Project Author	**Position**	Project Supervisor	**Position**	Project Supervisor

Contents

List of Figures

List of Tables

Chapter 1

State of the art

This chapter gives a brief introduction to the background of this project. The first thing that is discussed is classical methods of speech synthesis, as well as a brief mention of deep learning methods used in this particular task. We also discuss text classification.

By the end of this chapter, the necessary background will have been introduced for its application in the following chapters.

1.1 Speech synthesis

Speech synthesis is the process of generating a synthetic speech signal, emulating that of a human being. More specifically, this project focuses on synthesizing the signal given text level information. That means, mapping a piece of text to the sounds that a human would make. It also has to be possible to generate any text that is given to the system.

Speech synthesis can be applied to designing computer interfaces such as Apple's Siri or Amazon's Alexa. This section reviews some of the classical and more modern approaches.

1.1.1 Concatenative synthesis by unit selection

Unit selection systems render a speech signal by concatenating waveform fragments from a large single-speaker database [18]. Each unit in the database is also associated by a prosodic feature vector which contains information about pitch and duration of the single phonemes, that is used when finding the best concatenation of waveforms that matches a given target input (the phrase that is being synthesized).

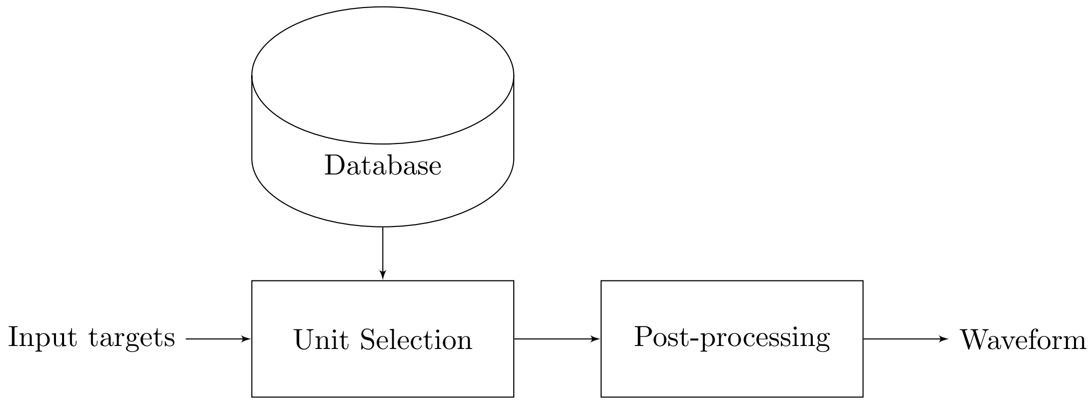


Figure 1.1: Concatenation system scheme.

Figure 1.1 shows the concatenative scheme. The system is composed of two modules, the first is the unit selection module where the best sequence of waveforms that matches the desired input prosodic targets is selected from the database. This sequence is obtained by minimizing the following expression using the Viterbi algorithm:

$$\hat{\mathbf{u}}_1^N = \underset{\mathbf{u_1^N}}{argmin} \left\{ \rho \sum_{i=1}^{N} c_{target}(u_i, t_i) + (1-\rho) \sum_{i=2}^{N} c_{concat}(u_{i-1}, u_i) \right\} \tag{1.1}$$

Where $\hat{\mathbf{u}}_\mathbf{1}^\mathbf{N}$ is the best selection of the u_i units in the database and t_i is the i-th target input of the system. c_{target} is the cost of using a particular unit to match a wanted prosodic target, and c_{concat} is the cost of concatenation of two units. The expression has a weighting term ρ that can be used to adjust the outcome.

The post-processing is used to fix the prosodic discontinuities of the raw selected units using pitch-synchronous overlap-add (PSOLA) [26], either in the time domain (TD-PSOLA) or the frequency domain (FD-PSOLA). This module is needed specially when the volume of the phoneme database is not very big and the chances of having discontinuities is high.

The outcome of concatenative systems can be highly natural-sounding speech, but it has a drawback: the database is always needed therefore the memory footprint of this technique can be high. To remedy this, statistical approaches are used as seen in the following section.

1.1.2 Statistical Parametric Speech Synthesis

Concatenative systems suffer from a large memory footprint due to the requirement of a waveform fragment database. To remedy this, statistical parametric speech synthesis (SPSS) is based on modeling the statistical properties of speech. To explain this approach, the scheme from Figure 1.3 from the HTS framework [38] is used as a guide.

This technique is perfomed in two phases. The first one is a training phase (top half in Figure 1.3) where each phoneme in a language is modeled by a finite-state structure called Hidden Markov Model (HMM) using a speech corpus, and the second phase is the synthesis (bottom half) where the trained models are used to synthesize the speech.

In the training phase, a large speech corpus is used. From this corpus we extract the excitation and spectral parameters (acoustic features) from the audio files and the phoneme representations of the transcripts (linguistic features or Labels in the figure). This data is used to train a model for each phoneme. The models are HMMs from Figure 1.2

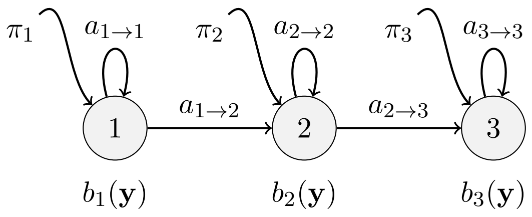


Figure 1.2: A 3-state left-to-right Hidden Markov Model.

Which can be represented by the touple $\lambda = \langle A, B, \Pi \rangle$ where A are the state transition probabilities $a_{i\to j}$, B are the acoustic observation distributions $b_i(y)$, and Π are the initial state probabilities π_i. The acoustic observations $\{y_1, y_2, y_3, ...\}$ are modeled by Gaussian Mixture Models (GMM). The parameters of the HMMs are estimated using the Baum-Welch algorithm [4].

In the synthesis phase, the trained models are used to generate the acoustic observations, also by means of ML, given set of linguistic features passed as inputs (text analysis block in Figure 1.3). The HMM acoustic observations are processed through a synthesis filter (generally a vocoder) to generate the final synthetic speech.

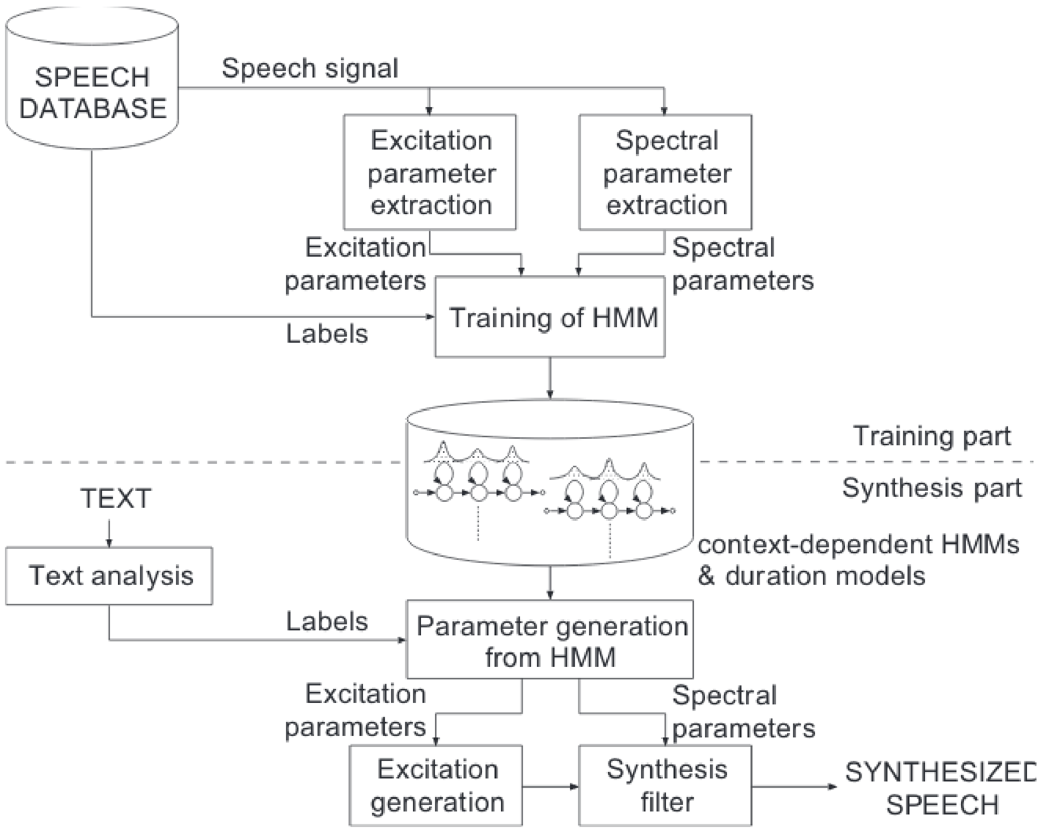


Figure 1.3: SPSS system based on HMMs. This scheme corresponds to the HTS framework [38].

SPSS systems are more efficient because they don't require a large database in the synthesis stage. Moreover, they also overcome another limitation of concatenative systems of not being able to generate multiple voices and styles of speech. SPSS offer this flexibility when the parameters are modified appropriately as described in [27] and [38].

1.1.3 Recurrent Neural Network-based Speech Synthesis

Another approach to speech synthesis more closely related to this project, is using recurrent neural networks (RNNs) to process a sequence of linguistic features and predict the same excitation and spectral parameters than SPSS.

Figure 1.4 from [9] is used as a guide for this section, but only the synthesis part, since the training will be covered later in this chapter. This one in particular uses a special kind of RNN called LSTM-RNN, also covered later in this chapter.

This approach is similar to the HMM based one from the previous section, but this one uses a special kind of neural networks called recurrent neural networks (more about them later in this chapter) to generate the acoustic observations that are used to reconstruct the speech signal. More specifically, it contains two models: a duration model, and an acoustic model.

The input of this system is also a phonetic representation of the text (text analysis and linguistic feature extraction in the figure). The model first predicts the duration of a frame to obtain a phoneme duration using the duration model (bottom half of the figure).

Afterwards, the acoustic model (another LSTM-RNN) is used to generate as many acoustic predictions as to match the predicted duration of the phoneme, using the same linguistic features and the predicted duration as inputs (note that the same linguistic features can be used multiple times in this stage).

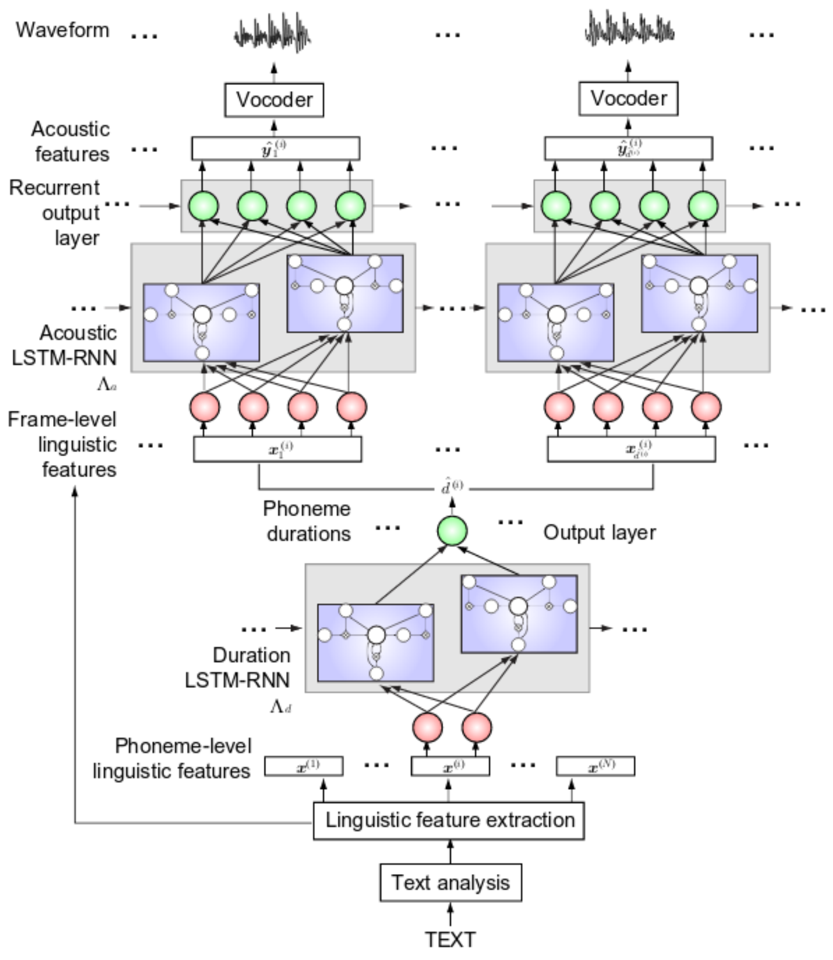
The predicted acoustic features are fed to a reconstruction filter or vocoder that generates the speech signal.

1.1.4 Expressive speech synthesis

One common limitation in the sythesis techniques explained above is that they might not be suitable for applications that benefit from expressive speech, such as audiobook reading. Since this project focuses on overcoming this limitation, this section reviews how this problem has been tackled in the past.

There exists several ways of obtaining expressive speech. One example involves defining an emotion target which tells the synthesizer how to generate the final waveform. In [7] and [14], different emotions are modeled using prosodic information such as pitch. They implement this in a concatenative system like the one in Section 1.1.1.

Another way of obtaining expressive speech in the bibliography was developed in the VEU research group [19]. In contrast with the previous examples, which involve manually



Waveform
Vocoder
Vocoder
Acoustic features
$\hat{y}_1^{(i)}$
$\hat{y}_{\hat{d}^{(i)}}^{(i)}$
Recurrent output layer
Acoustic LSTM-RNN Λ_a
Frame-level linguistic features
$x_1^{(i)}$
$x_{\hat{d}^{(i)}}^{(i)}$
$\hat{d}^{(i)}$
Phoneme durations
Output layer
Duration LSTM-RNN Λ_d
Phoneme-level linguistic features
$x^{(1)}$
$x^{(i)}$
$x^{(N)}$
Linguistic feature extraction
Text analysis
TEXT

Figure 1.4: RNN-based SPSS scheme. Concepts such as LSTM that appear in the figure are covered at later section of this chapter. The figure is originally from [9]

selecting a desired emotion target, this work uses deep neural networks (more about them this later in this chapter) to predict acoustic features of expressive speech from text, meaning the exressiveness is obtained from the text itself, something that can be used to develop synthesizers that have adaptive expressiveness.

1.2 Text classification and sentiment analysis

Text classification is a task in natural language processing [10] (NLP) where a text is assigned a label automatically.

A classic model used used in text classification is bag of words [33] (BOW), where a piece of text is assumed to be a sequence of words picked at random from a specific set. In the topic of text classification, classifying text using BOW would involve predicting the set the text belongs to, for example, the probability that an email is SPAM.

More recently, more modern techniques such as deep learning have been used to improve the accuracy of these systems. In [39] text is treated as a signal and processed using Convolutional neural networks (more on this in the next sections) and [20] uses vector representations of words (word embeddings [15]) to classify text.

One well known application of text classification, that is used in this project, is sentiment analysis (SA). SA can be applied to classification tasks such as the one from the Stanford Sentiment Treebank dataset [31], which contains a list of movie reviews and the goal is to predict how positive or negative they are based on the text.

1.3 Deep Learning

In recent years, deep structured learning, or more commonly called deep learning is a set of techniques that has risen in popularity and established itself as a new area of machine learning among the scientific and research communities. [13]

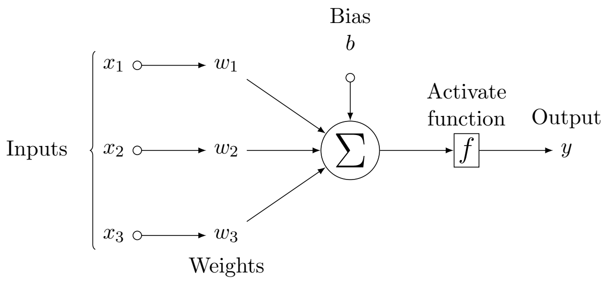
In [2] the authors discussed the limitations of shallow architectures and provided a theoretical basis to the advantages of deeper ones. But it has only been in recent years that such architectures have been able to be put into production with ever-growing popularity, thanks to high performance compute technologies being every time more available and affordable to consumers.

With deep learning architectures, it is possible to learn complex non-linear representations from large amounts of data. Raw data can be processed and turned into several levels of higher level representations to make it possible to reason about it and make tasks such as classification easier and more effective.

In addition to more computational power, large datasets that can take advantage of deep models such Imagenet [29] are being made publicly available. These datasets are often used as benchmarks and serve as points of reference to drive research and development forward.

1.3.1 Deep Neural Networks

In Deep Learning, a common architecture is the so called Deep Neural Network (DNN), which is an artificial neural network with a large number of hidden layers. The desire to decomposing a signal into higher levels of abstraction drives such neural networks configurations, and since more layers require more trainable parameters, large amounts of training data are also needed to train them.



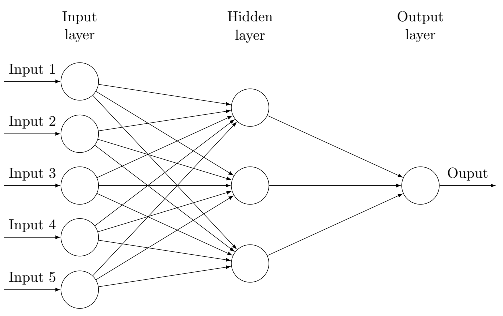


Figure 1.5: Basic neuron (left) and neural network configuration (right).

DNNs are defined by stacking several layers of basic units called neurons. In each layer, a linear operation takes place, where the inputs $\mathbf{x} = \{1, x_1, x_2, ..., x_N\}$ (1 is for a bias term) are linearly combined by a set of weights that are characteristic of each layer. The output of this linear operation is fed to a non-linear activation function (such as a sigmoid ((1.3)) or a tanh) which introduces the non-linearities of the system. Equation (1.2) shows the operation that takes place in the i-th layer, where $\mathbf{W_i}$ is the matrix of weights, $\mathbf{x_i}$ is the input vector of the layer, $\mathbf{y_i}$ is the output vector and σ is the activation function.

$$\mathbf{y_i} = \sigma\left(\mathbf{W_i} \cdot \mathbf{x_i}\right) \tag{1.2}$$

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1.3}$$

The outputs of this activation function are fed to the next layer of neurons until the last layer of a network, typically a softmax activation function for classification tasks or linear operation in regression ones. Figure 1.5 shows a neural network configuration.

1.3.2 Optimization

Optimizing, or training, a DNN is the process of finding the best possible set of weights that accomplish the best results in a given task. A task that DNNs are used for is classification, where a label is assigned to each input vector (predicting whether an image is from a cat or a dog) and also regression (predicting the expected cost of a property given the land size and proximity to the coast).

To optimize the network a cost or loss function is defined to measure the error of the predictions (such as the root mean square error or the cross-entropy error [16]). given a loss, we can improve the performance of a model by translating the weight vector in the direction of the gradient.

Equation (1.4) is a general loss function that is computed from a random batch B of vectors from a dataset $\mathbf{X}$ where $B \subset \mathbf{X}$.

$$L(\mathbf{X};\mathbf{w}) = \sum_{\mathbf{x}_k \in B} E(\mathbf{x_k}) \tag{1.4}$$

Where $\mathbf{w}$ are the weights of the model and $E(x_k)$ is the contribution of the k-th vector of the batch to the loss. This technique of not using the whole dataset to compute the loss is called stochastic gradient descent (SGD) [6].

Before updating the weights, the partial derivatives of the loss function are computed using the back-propagation algorithm [8]. For this we need to be able to compute the derivatives of both the loss function and the activation functions of the neurons. The update can be performed like this:

$$w_i[n+1] = w_i[n] - \lambda \frac{\partial L(\mathbf{x};\mathbf{w}[n])}{\partial w_i} \tag{1.5}$$

Where w_i is the i-th weight of the model and $\frac{\partial L(x;w[n])}{\partial w_i}$ is the partial derivative of the loss function (1.4) with respect to the i-th weight. $\mathbf{w}[n]$ are the weights of the model in the n-th iteration.

The norm of the gradient is scaled by a factor called learning rate (expressed as λ in Equation (1.5)) to speed-up or slow-down the convergence of the loss function during training.

There are more elaborated ways of performing these weight updates, such as the Adam [22] and Adadelta [37] optimizers, which are the ones used in this project.

1.4 Recurrent networks

Recurrent neural networks (RNN) are a special type of architecture capable of modeling sequences, where the outputs of a hidden layer are fed back to the inputs of the same layer. This feedback loop introduces a state in the neurons and the output of the layers can be rewritten as:

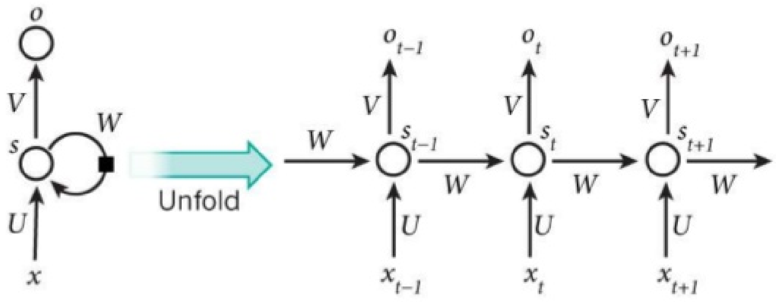


Figure 1.6: Unfolded RNN

$$\mathbf{y_t} = \sigma\left(\mathbf{W} \cdot \mathbf{x_t} + \mathbf{U} \cdot \mathbf{y_{t-1}}\right) \tag{1.6}$$

Where $\mathbf{x_t}$ is an input vector at the t-th timestep and $\mathbf{U}$ is an additional matrix of tranable weights. Because of the introduction of a neuron state, RNNs can model various types of sequential data (predict the next frame of a video given the N first or predict the duration of each of the phonemes in a sentence as seen previously).

Recurrent networks can still be trained efficiently by using back-propagation through time [34] which is specific case of back-propagation where the errors are also back-propagated back in time. If we expand (1.6) we get:

$$\mathbf{y_t} = \sigma\left(\mathbf{W} \cdot \mathbf{x_t} + \mathbf{U} \cdot \sigma\left(\mathbf{W} \cdot \mathbf{x_{t-1}} + \mathbf{U} \cdot \sigma\left(\cdots \sigma\left(\cdots\right) \cdots\right)\right)\right) \tag{1.7}$$

This expansion is graphically represented in Figure 1.6. The recursive multiplication of U_t make RNNs a special case of DNNs, where the repeated multiplications can cause the gradients to become too small or too big by the time they reach the inputs of the network when doing back-propagation. This is known as vanishing or exploding gradient problem [3], a phenomenon that occurs in SGD.

To mitigate this problem in RNNs, we use specially designed recurrent neurons such as Long Short-Term Memory [17] (LSTM) as opposed to regular RNN neurons. These cells work with a more sophisticated mechanism that controls the flow of information coming in and out of the cells. LSTMs have proven to be effective at modeling long-term sequences and that's the reason why they are used in this project.

1.5 Convolutional Neural Networks

As seen in Equation (1.2), a neuron activation depends on all the output activation of the previous layer. Convolutional neural networks (CNN) are a type of architecture that contains a type of hidden layers called Convolutional layers. These architectures have proven to be very effective in computer vision applications but they are also used with success in NLP tasks [39].

Rather than combining all the outputs of the previous layer, CNN neurons only focus on the activations of a fixed number of neurons from the previous layer [23]. While each neuron focuses on a different set of inputs, the total number is fixed and the weights are shared among the activations of the layer. Because of this, we can think of these layers as having an associated kernel that is used to filter the previous layer by means of a convolution to produce a feature map in the layer. The number of feature maps is a parameter of the convolutional layer.

Once the state of the art and the deep learning concepts that are used throughout this project have been reviewed, the next implementation chapters can be explained and discussed.

Chapter 2

Baseline speech synthesis

2.1 Baseline development

The first stage of this project was to develop a baseline speech synthesizer. This model is used as a reference when comparing the results of the experiments explained in the evaluation chapter. The architecture (Figure 2.1) is based on the work done by [27] by using the Socrates Text-to-speech framework developed in the VEU research group at UPC, which is based on the Keras deep learning library. This is a RNN-LSTM based model, which, as mentioned in section 1.1.3, contains a duration model and an acoustic model which are trained independently.

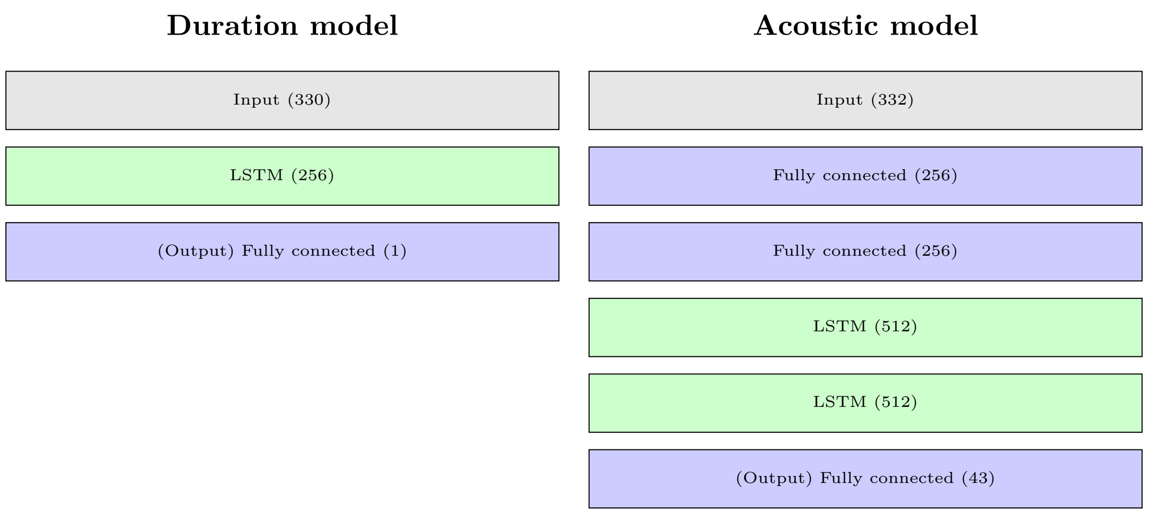


Figure 2.1: Baseline models.

- The duration model predicts the duration of a phoneme. This model takes a vector of linguistic features with information about the phoneme and its context and outputs a single value that is the log-compressed duration of the phoneme (this log-compression is explained in the data preparation section).
- The acoustic model predicts the excitation and spectral parameters of a frame of speech.

The input of this system is the same vector of linguistic features, the normalized duration of the predicted value from the previous model and the relative position of the frame within the duration of the phoneme. The output of this model is also explained in the next section.

2.2 Data preparation

The corpus that has been used for this project contains approximately 20 hours or audiobooks along with the transcripts. This corpus was originally produced for the Blizzard Challenge 2013 [21] and is already segmented at the utterance level, removing the need to align the whole audio stream with the raw text data. The reason we chose this corpus is because of its richness in expressive content. Table 2.1 shows some information about the audio files that we get from it.

Metric	Value
Sampling rate (Hz)	16000
Bit depth (bits)	16
Channels	mono
Length (seconds mean)	7.23
Length (seconds std)	4.52
Speakers	1

Table 2.1: Information about the Blizzard Challenge utterances.

2.3 Obtaining Acoustic features

As mentioned in section 1.1.3, this RNN based speech synthesizer is based on SPSS and as such it doesn't output the waveforms directly but the excitation and spectral parameters before they are reconstructed by a vocoder. The vocoder that we used is Ahocoder [12].

Every audio file is framed by means of a sliding window with a stride of 5 milliseconds. Ahocoder then processes each frame to produce a set of excitation and spectral parameters which are:

- Mel Frequency Cepstral Coefficients (MFCC) of order $p = 39$, which corresponds to 40 coefficients. This represents the information from the speech formants.

- Pitch contour ($log(F_0)$). Unvoiced frames correspond to $F_0 = 0$. Ahocoder outputs a value of -10^8 when the frame is unvoiced.

- Maximum voiced frequency (fv), This value is 0 when a frame is unvoiced.

- Voiced/Unvoiced (UV) flag. This value indicates if a given frame of audio corresponds to a voiced or unvoiced sound and is obtained using the $log(F_0)$ output of Ahocoder.

To perform this data extraction from the Blizzard corpus, the Python and Bash scripting languages were used to parallelize the whole process. The basic unit of work of this step is to spawn an Ahocoder process with the path of the file, and save the speech parameters that it generates to disk. Because this operation is single threaded, and because of the large amount of data that we had in the Blizzard Challenge dataset (20 hours of speech), this process was parallelized in order to fully utilize the computing resources of the server. This involved writing two scripts:

- **split_names.py** is a Python script that takes a list of files (All the filenames from the Blizzard corpus) and splits it into smaller chunks.

- **process_ahocode.sh** is a Bash script that reads a list of files and processes each of them through Ahocode and saves the speech parameters to the disk.

The filenames of the corpus was split into 30 chunks (the server has 32 CPUs) using the **split_names.py** script and each of the chunks was processed by **process_ahocode.sh**. Using this method inspired by [27] this whole data preparation process was accomplished in less than one hour fully utilizing the system resources.

2.4 Acoustic feature normalization

The data is not used as it is at the output of the Ahocoder directly. These acoustic predictors are normalized before they are used to train the acoustic model for a good behavior of the back-propagation algorithm, as discussed in [27].

The outputs of the Ahocoding process ($MFCC$, $log(f_0)$ and f_v) were normalized so that they were bound between a minimum and a maximum value range. This range is chosen to be $0.01, 0.99$. The normalization of the acoustic outputs is shown in equation 2.1

$$\hat{y} = 0.01 + (0.99 - 0.01)\frac{y - y_{min}}{y_{max} - y_{min}} \tag{2.1}$$

This doesn't work for the $log(F_0)$ features however. Because Ahocoder outputs a value of -10^8 when a frame is not voiced, equation (2.1) would compress the values from the voiced frames too much. [27] solves it by keeping the values from the voiced frames and interpolating the values in the unvoiced regions as shown in Equation 2.2:

$$logF_0^i = logF_0^p + (logF_0^n - logF_0^p) \cdot \frac{i - p}{n - p} \tag{2.2}$$

Where n is the next voiced frame's frame index, F_0^n is the next voiced frame's first value, p is the previous voiced value's frame index, F_0^p is the previous voiced value and F_0^i is the i-th new interpolated value to be replaced in the original output of the Ahocoding process (Figure 2.2 has an example of this interpolation). We can then use the UV flag to recover the original format after denormalization and reconstruct the waveform.

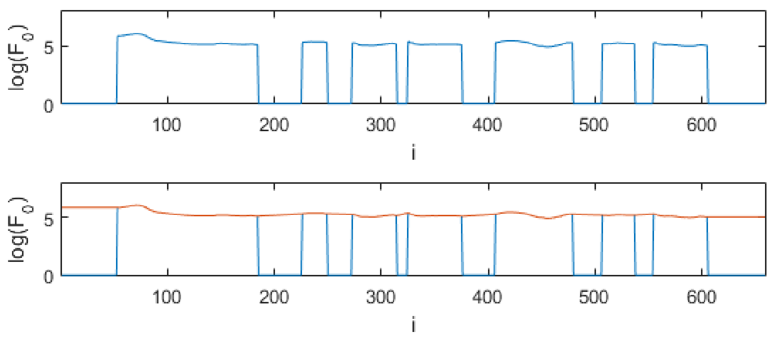


Figure 2.2: F_0 contour interpolation

These acoustic features are used to train the acoustic model of the system. The data is structured for training as a series of input-output vector pairs laid out in the following manner in the training data tables:

```
<linguistic features 1, duration 1> <acoustic features 1>
<linguistic features 1, duration 1> <acoustic features 2>
<linguistic features 1, duration 1> <acoustic features 3>
<linguistic features 2, duration 2> <acoustic features 4>
<linguistic features 2, duration 2> <acoustic features 5>
<linguistic features 2, duration 2> <acoustic features 6>
<linguistic features 2, duration 2> <acoustic features 7>
<linguistic features 3, duration 3> <acoustic features 8>
<linguistic features 3, duration 3> <acoustic features 9>
...
```

Figure 2.3: Training table used to train the acoustic model is stored in the server disk.

Which are the examples that are used to train the acoustic model. Some of the inputs are repeated depending on the duration of the phonemes (the linguistic features are explained in the next section). To create these tables, another python script **gen_aco_tables.py** was written for the project. This script reads a list of files corresponding to the single sentences from the Blizzard corpus and does the following:

1. Reads the file of linguistic features (explained in the next section) containing both the phonetic segmentation and the phoneme durations.

2. Reads the corresponding acoustic features from Ahocoder (obtained in the previous section) and writes a line containing: the vector of linguistic features, the log normalized duration of the phoneme, the relative position of the ahocoder window within the current phoneme, and the vector of acoustic features ($MFCC$, $log(f_0)$, f_v and UV). This line in repeated for as many windows the phoneme lasts (modifying the relative duration value every line). At the end there might be a remainder duration error because the number of windows the phoneme lasts might not be an integer number. This remainder is carried to the next phoneme.

3. Once all the data has been collected, the values are normalized and written to disk for use in Socrates.

We execute this script for both the training data, the test data and the validation data. These can be done in parallel but it is not as critical.

2.5 Obtaining linguistic features

Each of the audio transcriptions was transformed into a lower level phonetic representation called Label that contains a set of both real and categorical descriptors with information about each phoneme and its context within a sentence. Table 2.2 contains a description of the values that are included in this representation.

When a phoneme is converted into a Label, it has the following format:

```
p1ˆp2$−$p3$+$p4$=$p5˜p6_p7/A:a1_a2_a3/B:b1−b2−b3˜b4−b5&b6−b7#b8−b9$b10 ...
−b11!b12−b13;b14−b15|b16/C:c1+c2+c3/D:d1_d2/E:e1+e2˜e3+e4&e5+e6#e7+e8 ...
/F:f1_f2/G:g1_g2/H:h1=h2˜h3=h4|h4/I:i1_i2/J:j1+j2−j3
```

label format

Symbol	Description
p1	phoneme identity before the previous phoneme
p2	previous phoneme identity
p3*	current phoneme identity
p4*	next phoneme identity
p5*	the phoneme after the next phoneme identity
p6*	position of the current phoneme identity in the current syllable (forward)
p7*	position of the current phoneme identity in the current syllable (backward)
a1	whether the previous syllable is stressed or not (0; not, 1: yes)
a2	whether the previous syllable is accented or not (0; not, 1: yes)
a3	number of phonemes in the previous syllable
b1*	whether the current syllable stressed or not (0: not, 1: yes)
b2*	whether the current syllable accented or not (0: not, 1: yes)
b3*	the number of phonemes in the current syllable
b4*	position of the current syllable in the current word (forward)
b5*	position of the current syllable in the current word (backward)
b6*	position of the current syllable in the current phrase(forward)
b7*	position of the current syllable in the current phrase(backward)
b8*	number of stressed syllables before the current syllable in the current phrase
b9*	number of stressed syllables after the current syllable in the current phrase
b10*	number of accented syllables before the current syllable in the current phrase
b11*	number of accented syllables after the current syllable in the current phrase
b12*	number of syllables from the previous stressed syllable to the current syllable
b13*	number of syllables from the current syllable to the next stressed syllable
b14*	number of syllables from the previous accented syllable to the current syllable
b15*	number of syllables from the current syllable to the next accented syllable
b16*	name of the vowel of the current syllable

c1*	whether the next syllable stressed or not (0: not, 1:yes)
c2*	whether the next syllable accented or not (0: not, 1:yes)
c3*	the number of phonemes in the next syllable
d1	gpos (guess part-of-speech) of the previous word
d2	number of syllables in the previous word
e1*	gpos (guess part-of-speech) of the current word
e2*	number of syllables in the current word
e3*	position of current word in the current phrase (forward)
e4*	position of current word in the current phrase (backward)
e5*	number of content words before the current word in the current phrase
e6*	number of content words after the current word in the current phrase
e7*	number of words from the previous content word to the current word
e8*	number of words from the current word to the next content word
f1*	gpos (guess part-of-speech) of the next word
f2	number of syllables in the previous word
g1	number of syllables in the previous phrase
g2	number of words in the previous phrase
h1*	number of syllables in the current phrase
h2*	number of words in the current phrase
h3*	position of the current phrase in utterance (forward)
h4*	position of the current phrase in utterance (backward)
h5*	Phrase modality (question, exclamation, etc.)
i1*	number of syllables in the next phrase
i2	number of words in the previous phrase
j1*	number of syllables in this utterance
j2*	number of words in this utterance
j3*	number of phrases in this utterance

Table 2.2: Label format. The symbols tagged with an asterisk reference the future context of the phoneme.

A full example of this conversion is shown in Figure 2.4:

To obtain the Labels of the Blizzard corpus, first the sentences were phonetically segmented. This process was done using the Ramses and Ogmios [5] software. The whole process was mostly automated following a guide:

1. The text files are converted to their phonetic representation. Speech parameters are also computed from the audio files from where a codebook is created and quantized.

2. The next step estimates a semi-continuous HMM, where only the GMM parameters are

```
pau^D-i+O:=t~2_1/A:0_0_2/B:0-0-2~1-1&1-10#0-0$0-2!0-0;3-4|i/C:0+0+1/D:NN_2/E:DT+1~1+5&0+3#0+1/F:...
D^i-O:+t=@~1_1/A:0_0_2/B:0-0-1~1-6&2-9#0-0$0-2!0-0;4-3|O:/C:0+0+2/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN...
i^O:-t+@=b~1_2/A:0_0_1/B:0-0-2~2-5&3-8#0-0$0-2!0-0;5-2|@/C:0+0+2/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN_...
O:^t-@+b=aI~2_1/A:0_0_1/B:0-0-2~2-5&3-8#0-0$0-2!0-0;5-2|@/C:0+0+2/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN...
t^@-b+aI=Q~1_2/A:0_0_2/B:0-0-2~3-4&4-7#0-0$0-2!0-0;6-1|aI/C:0+1+1/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN...
@^b-aI+Q=g~2_1/A:0_0_2/B:0-0-2~3-4&4-7#0-0$0-2!0-0;6-1|aI/C:0+1+1/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN...
b^aI-Q+g=r~1_1/A:0_0_2/B:0-1-1~4-3&5-6#0-0$0-1!0-0;7-5|Q/C:0+0+3/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN_...
aI^Q-g+r=@~1_3/A:0_1_1/B:0-0-3~5-2&6-5#0-0$1-1!0-0;1-4|@/C:0+0+2/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN_...
Q^g-r+@=f~2_2/A:0_1_1/B:0-0-3~5-2&6-5#0-0$1-1!0-0;1-4|@/C:0+0+2/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN_1...
g^r-@+f=i~3_1/A:0_1_1/B:0-0-3~5-2&6-5#0-0$1-1!0-0;1-4|@/C:0+0+2/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN_1...
r^@-f+i=@~1_2/A:0_0_3/B:0-0-2~6-1&7-4#0-0$1-1!0-0;2-3|i/C:0+0+2/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN_1...
@^f-i+@=v~2_1/A:0_0_3/B:0-0-2~6-1&7-4#0-0$1-1!0-0;2-3|i/C:0+0+2/D:DT_1/E:NN+6~2+4&0+3#1+1/F:IN_1...
f^i-@+v=@~1_2/A:0_0_2/B:0-0-2~1-1&8-3#0-0$1-1!0-0;3-2|@/C:0+0+1/D:NN_6/E:IN+1~3+3&1+2#1+2/F:DT_1...
i^@-v+@=h~2_1/A:0_0_2/B:0-0-2~1-1&8-3#0-0$1-1!0-0;3-2|@/C:0+0+1/D:NN_6/E:IN+1~3+3&1+2#1+2/F:DT_1...
@^v-@+h=O:~1_1/A:0_0_2/B:0-0-1~1-1&9-2#0-0$1-1!0-0;4-1|@/C:0+1+3/D:IN_1/E:DT+1~4+2&2+1#1+1/F:NN_1...
v^@-h+O:=s~1_3/A:0_0_1/B:0-1-3~1-1&10-1#0-0$1-0!0-0;5-0|O:/C:0+0+0/D:DT_1/E:NN+1~5+1&2+1#2+0/F:SI...
@^h-O:+s=pau~2_2/A:0_0_1/B:0-1-3~1-1&10-1#0-0$1-0!0-0;5-0|O:/C:0+0+0/D:DT_1/E:NN+1~5+1&2+1#2+0/F:...
h^O:-s+pau=_~3_1/A:0_0_1/B:0-1-3~1-1&10-1#0-0$1-0!0-0;5-0|O:/C:0+0+0/D:DT_1/E:NN+1~5+1&2+1#2+0/F:...
O:^s-pau+_=_~1_1/A:0_1_3/B:0-0-1~1-1&0-11#0-0$0-2!0-0;1-0|_/C:0+0+0/D:NN_1/E:SIL+1~0+6&0+3#1+0/F:...
```

Figure 2.4: Truncated label representation of the phrase "the autobiography of a horse."

quantized, for every phonem using the speech parameters. Using these HMM models and the phonetic transcriptions, a force alignment is performed to find the transitions of the phonemes and the internal pauses and silences in the sentences using the Viterbi algorithm.

3. Next it creates a diphone transcription [24] of the sentences and estimates an HMM of them.

4. Finally, a force alignment is performed again to obtain the diphone transitions in order to compute the duration of each phoneme.

5. A script is called to convert the outputs from the Ogmios program to the Label representation with the phoneme durations.

2.6 Linguistic features normalization

The linguistic features, also had to be normalized prior to training. As seen previously, the label format contains both categorical features and real features. The categorical information was encoded so that all categories are orthogonal to each other by using a one-hot encoding. As a small example of a one-hot encoding, consider the three categories $\{A, B, C\}$. The one-hot encoding for the three categories is $\{(1,0,0),(0,1,0),(0,0,1)\}$.

The real features from the acoustic features are normalized so that the mean equals zero and the standard deviation equals one for each of them. This operation is called z-norm and it is shown in 2.3.

$$z = \frac{x - \mu}{\sigma} \tag{2.3}$$

The predicted duration of each phoneme was log-compressed to avoid having a long-tailed distribution that can distort the training of the model when using the Mean Square Error (MSE) minimization as discussed in [27]. Figure 2.5 contains the histogram of the raw predicted durations and the histogram after the log-compression operation. This operation was performed by simply taking the natural logarithm of the stored phoneme durations:

$$\hat{d} = log(d) \tag{2.4}$$

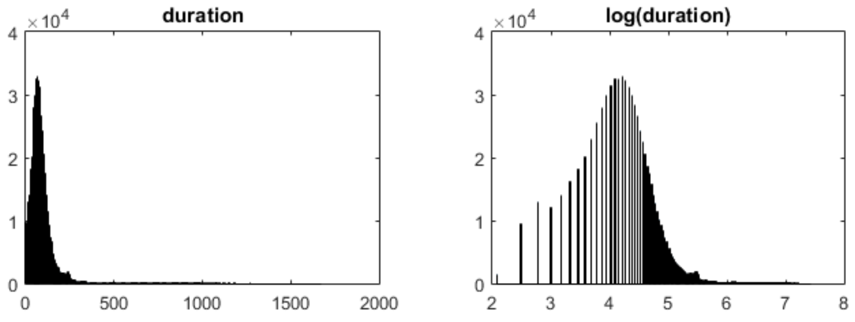


Figure 2.5: Histogram of the phoneme duration predictions by Ogmios.

Once we have obtained the linguistic features, we discard of those who reference the past and only keep the ones tagged with an asterisk ($*$) in Table 2.2. This is because the models are based on LSTM-RNN which are known to be good at modeling long term dependencies from the past[17].

2.7 Training of the baseline

As mentioned before, the system is developed within the Socrates Test-to-speech framework. In the version that was used, this program allows us to define the full architecture in a plain-text configuration file. In this file we specify the number of layers, inputs parameters, the paths where the training, test and validation data are stored in the disk, and the path where to store the intermediate files that the framework generates during training (network weights, loss curves and cached information). An example of configuration file is shown in Figure 2.6.

```
[[tts_core]]
name="labdecode_core"

[[tts_core]]
name="dur_rnn_core"
topology="baseline/dur_rnn_core-eng.cfg"
dur_stats="model/73/dur/dur.stats.train"
train_table="../tables/duration/baseline/train.dur"
test_table="../tables/duration/baseline/test.dur"
valid_table="../tables/duration/baseline/valid.dur"
loss_path="baseline/loss_curves/"
tables_cache_path="baseline/tables_cache"

[[tts_core]]
name="dur2aco_norm_core"
dur_stats="model/73/dur/dur.stats.train"

[[tts_core]]
name="aco_rnn_core"
dur_stats="model/73/aco/dur.stats.train"
aco_stats="model/73/aco/aco.stats.train"
train_table="../tables/acoustic/baseline/train.aco"
test_table="../tables/acoustic/baseline/test.aco"
valid_table="../tables/acoustic/baseline/valid.aco"
topology="baseline/aco_rnn_core-eng.cfg"
tables_cache_path="baseline/tables_cache"
loss_path="baseline/loss_curves/"

[[tts_core]]
name="ahodecode_core
```

Figure 2.6: Configuration example for the baseline system. Some options have been omitted.

In this example, we specify that this model is composed of five modules, called cores within the framework. Each of these cores perform the following actions

1. **labdecode_core** decodes the label format from Section 2.5 and converts it to the input that the model expects (converts the categories to one-hot and normalizes the numeric values)

2. **dur_rnn_core** predicts the duration of the phoneme that was provided from the pre-

vious core.

3. **dur2aco_norm_core** denormalizes the predicted duration and generates the input features that are feed to the acoustic model. It also adds the durations and relative durations as mentioned at the beginning of this chapter.

4. **aco_rnn_core** predicts the acoustic features explained in Section 2.3.

5. **ahodecode_core** takes the output from the previous core, formats the data to be input to the Ahodecoder, and produces a WAVE file containing the synthesized speech.

The configuration file also has access to the "stats" of the features that contain the minimum and maximum values so that the compression from 2.1 can be undone.

As mentioned previously, the Blizzard Challenge corpus contains roughly twenty hours of speech data. Due to complications loading the data into memory using the framework, we had to cut the total amount down to 30% to train the acoustic model. Newer releases of this framework deal with this problem so it can handle the full corpus data.

This chapter has covered the baseline architecture, data preparation and training. The next step of the project was to add expressive features to the system. The following chapter focuses on this.

Chapter 3

Expressive speech synthesizer

Once we had the system up and running it was time to introduce the modifications that would allow for expressive speech to be modeled by the system. The way this is done in this project is by introducing a new set of features as inputs to the system (both the duration and acoustic model). These are known as the embeddings that encapsulate the information about the way a given sentence is supposed to be synthesized like, given its semantic content, and they are used alongside the linguistic features from the baseline system. In this section we describe the process of obtaining these new features. Other than these new inputs, the developed system shares the same architecture from the baseline system.

3.1 Sentiment analysis & embedding extraction task

Once the baseline system was developed, the additions to the initial system were developed. These involve obtaining the expressive paragraph embeddings that would allow for the modeling of the expressiveness of the speech signal. We used sentiment analysis to capture the expressive information of the Blizzard Challenge dataset that was used in the baseline system. Because the sentences on this dataset are not made for sentiment analysis tasks, they lack the information to train a text classification system. To perform this text classification task, we used the Stanford sentiment treebank dataset [31], which contains a list of movie reviews. An example of a tagged sentence from this dataset is shown in figure 3.1.

Each of the nodes on the tree is tagged in a scale from 0 to 4, 0 meaning very negative and 4 being a very positive text. Each sub-sentence was tagged down to the single words of the sentence. We used this dataset to train a CNN-based classifier similar to [20]. This architecture is shown in Figure 3.2.

```
(2 (3 (3 Effective) (2 but)) (1 (1 too−tepid) (2 biopic)))

Tag Sentence
2   Effective but too−tepid biopic
2   Effective but
3   Effective
2   but
1   too−tepid biopic
1   too−tepid
2   biopic
```

Figure 3.1: Example of sentence tree corresponding to the sentence ”Effective but too-tepid biopic”

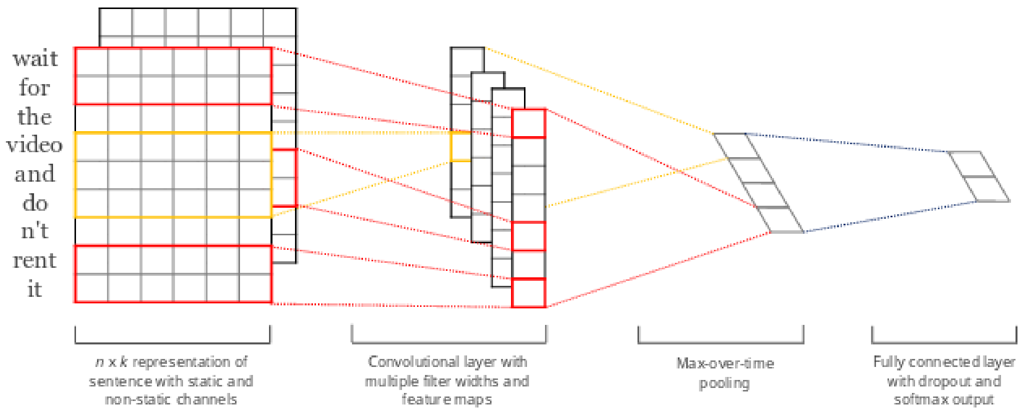

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 3.2: Text classification architecture. The figure is originally from [20]

- The first layer is an embedding layer. Each word is transformed into a low dimensional (300) representation. This step converts a sentence into a signal suitable for the next convolutional layer. The embedding layer consists of a (N+1)*300 matrix where N is the size of the vocabulary and the n-th vector column corresponds to the embedding of the n-th word in the vocabulary. An extra column is used for words that are not from the vocabulary and contains a random value. This architecture in particular has two-channel inputs, one with trainable embeddings, and the other with fixed ones. Both embedding matrices are initialized to the Glove embeddings [28].

- A convolutional layer with filters of sizes 3, 4 and 5 with 100 feature maps each.

- A max-over-time pooling operation [11], where we select the maximum value from each of the feature maps.

- A Fully connected layer with softmax activations that give the output of the classification task. This layer is regularized using the dropout method with dropout probability of 0.5 [32] and the weights of this network are constrained using l2 regularization.

We choose the max-over-time pooling layer activations to obtain the additional inputs that added to the baseline synthesizer model. There is an issue with using the Stanford treebank dataset to train this system. Because the dataset is based on movie reviews, the text domain doesn't belong to the same from the Blizzard challenge data, which is the one we use to train the synthesizer models. Therefore, the sentiment analysis network has to be adapted. The following section explains the methodology that was followed in order to attempt to achieve this task.

3.2 Network domain adaptation

To perform the domain adaptation, we do the same classification task on a different architecture using the Blizzard challenge data. This network makes the prediction with the audio files as inputs. As explained before, this dataset is not prepared for text classification therefore they lack the class tags needed. We obtain these by using the network from the previous section after it has been trained using the Stanford sentiment treebank dataset. In [1] they train a waveform classifier using unlabeled audio files and a pre-trained network to obtain them from the video frames. We perform the same task using audio and text data instead.

This task is performed in two steps:

1. We train the audio based network by using the text network to obtain the labels. Because the labels that we obtain for the Blizzard sentences are probabilities, we optimize the network using KL-divergence using standard back-propagation [1].

2. After a fixed number of iterations, we fine tune [36] the text network by using labels obtained from the audio network. At this point, the two networks take turns to perform a back-propagation pass. Because the state of the weight of each network is constantly

changing, we intend to leak information about the audio files into the text network.

3.3 Training embeddings extraction and expressive synthesis

Sentiment analysis task

In order to train the embedding extraction, the Keras deep learning framework was used to define the two networks. First, the data had to be processed in order to train the system. As shown in Figure 3.1, the sentences in the Stanford sentiment treebank are provided by a tree representation. Each of these trees were traversed in order to collect all the sentences and sub-sentences of the dataset. Figure 3.3 shows the distribution of sentence lengths:

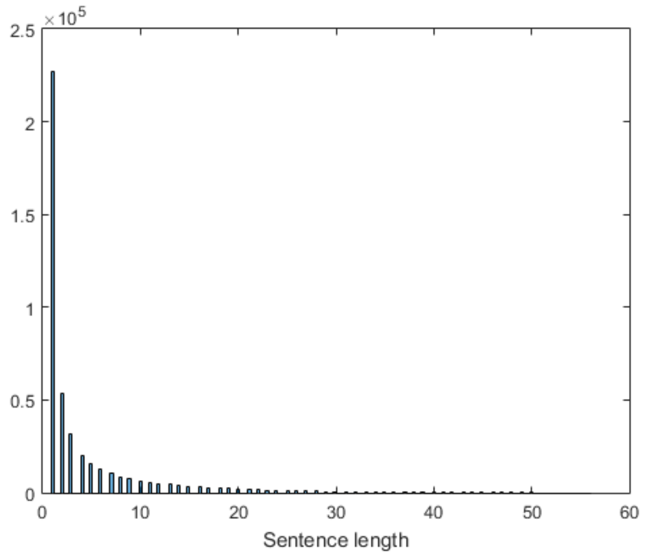


Figure 3.3: Distribution of sentence lengths in the Stanford sentiment treebank dataset.

To train this sentiment analysis task, we use all the sentences from the training dataset that are ≥ 3 in length (words). The test data however, only contains the root nodes of the sentences like they do in [20] and in the original paper from the dataset.

	conv	pool	conv	pool	conv	pool	conv	conv	conv
Feature maps	32	32	64	64	128	128	256	512	5
Filter size	64	8	32	8	16	8	8	4	5
Stride	2	2	2	2	4	2	4	1	1

Table 3.1: CNN used to process the Blizzard waveforms.

Domain adaptation task

Once the sentiment analysis task is done, the same network has to be adapter for the aforementioned reasons. To perform this task, we define a CNN from Table 3.1. The reason for choosing a CNN architecture as opposed to a different one such a DNN is because of the high dimensionality of the input. As aforementioned, this network takes whole waveforms from the Blizzard corpus. Convolutional layers have less trainable parameters since the number only depends on the size of the filters and the number of feature maps. This network also contains pooling layers which reduce the dimensionality of the input [30] and PReLU activations [35] which are activations with trainable parameters. We then use the following methodology:

1. The audio network is trained using the Blizzard challenge WAVE files. The label of a file is obtained by using the sentiment analysis network with the transcript. Perform a fixed number of back propagation passes to initialize the weights of the network. From this point, the networks will take turns in performing back-propagation steps.

2. We perform a back-propagation in the text network using a batch of sentences from the Blizzard corpus. The labels are obtained like in step 1 using audio network to obtain the labels.

3. We perform a back-propagation in the audio network using a batch of wave files from the Blizzard corpus. Again, use the text network to obtain the labels.

4. repeat step 2 for a fixed number of iterations.

3.4 Training the expressive speech synthesizer

Once the expressive embeddings are extracted for every sentence in the Blizzard Challenge corpus, we include the new inputs to both the duration and the acoustic model from the baseline system. This is specified in a new Socrates configuration file. Every linguistic feature vector that is input to the system includes the additional features that correspond to the full sentence the given phonemes belong to. After that, the process of training the models is the same that was followed in the baseline system.

One last processing step is to normalize the embeddings by compressing them between the $[0, 1]$ range.

The next chapter shows the evaluation results of the project.

Chapter 4

Evaluation & Results

In this chapter we explained how the evaluation of the developed tasks and systems was done.

4.1 Sentiment analysis and domain adaptation

As mentioned previously, the sentiment analysis task was trained using a dataset from Stanford. To perform this task we used the same test and train splits used in [31] which is the same from [20]. During training, both the loss and the prediction accuracy (from a five class prediction, as sen in the previous chapter) of this classification task were recorded (Figure 4.1) but only the loss is used in back-propagation.

The loss function was chosen among the ones available in the Keras framework to be the categorical cross-entropy [16] and the optimizer was Adadelta [37]. The batch size was set to 50 to stay as close as possible to the experiment conditions from [20].

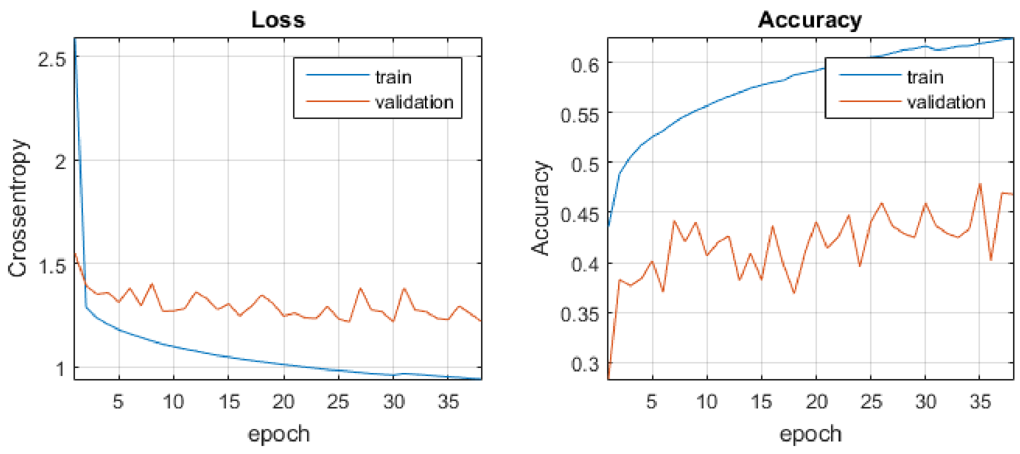


Figure 4.1: Training and validation curves of the sentiment analysis task.

We obtain 47.9% accuracy in this task. In [20] they obtain 47.4% performing the same task with a very similar architecture. Looking at the curves from Figure 4.1, it seems like the model could still be trained for a few more epochs.

The learned function from this model is transferred to the neural network from Table 3.1 using the KL loss as explained in the previous chapter. The optimizer is Adam. The evolution of the KL can be seen in Figure 4.2.

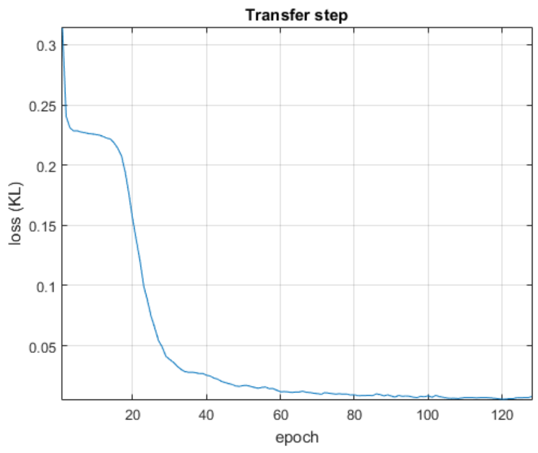


Figure 4.2: Evolution of the KL loss.

From Figure 4.2 we see how the Adam optimizer escapes a local minima. The adaptation from Section 3.3 step is performed afterwards, with the weights initialized from this transfer operation. This adaptation is shown in Figure 4.3

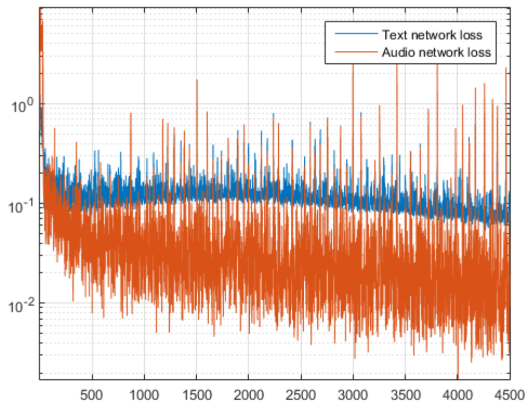


Figure 4.3: Evolution of the loss function for both the sentiment analysis network (text network in the figure) and the audio based one (audio network).

Figure 4.3 shows that the loss of the adapted sentiment analysis network (Text network in the figure) drops at the beginning but it continues to grow. Despite that, we keep the weights from the 100-th iteration to obtain the adapted embeddings.

4.2 Speech synthesis evaluation

This section contains both the objective and subjective evaluation results and how they were performed. A total of three experiments were evaluated. These experiments are:

1. Baseline system (only linguistic features from section 2.5)

2. Embeddings without doing the domain adaptation.

3. Embeddings but with the domain adaptation.

4.2.1 Objective evaluation

To perform the objective evaluation the Socrates framework keeps track of the loss evolution during the training. At the end of each batch, it saves the batch loss and at the end of each epoch, the validation data is used to obtain the validation loss of the models.

The metric that are computed by Socrates are the root mean square error (RMSE) for the duration predictions, mel cepstral distortion [25] (MCE) for the MFCC predictions, RMSE for the F_0 predictions and accuracy (%) for the UV flag.

	Duration	MFCC	F_0	UV
Random weights	152.75	22.35	58.50	34.31
Baseline	49.17	7.18	37.67	95.32
Expressive embeddings	51.76	7.35	41.46	94.72
Adapted embeddings	49.37	8.24	42.04	66.28

Table 4.1: Objective metrics.

Both the duration and acoustic models were trained minimizing the mean square error (MSE) using the Adam [22] optimizer. Figure 4.4 contains the training curves of the duration model and Figure 4.5 the acoustic model. We see that the baseline system is the one that reaches the lowest loss on both models. The acoustic model shows that the adapted embeddings are the worst in comparison.

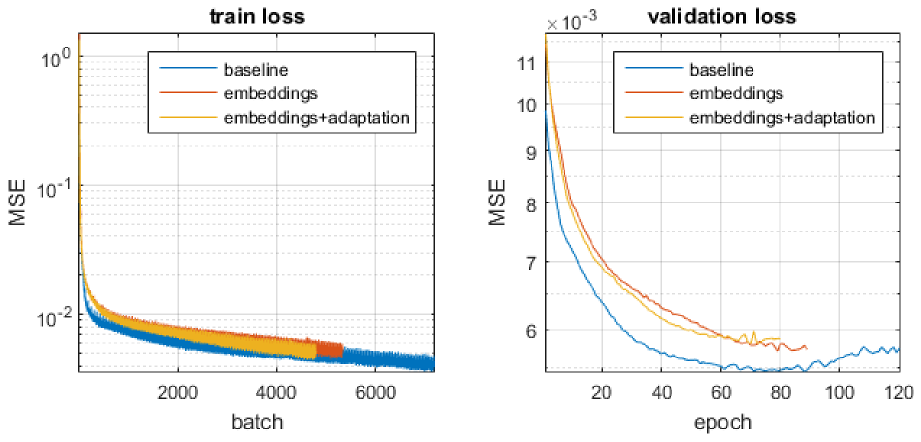


Figure 4.4: Duration model's training curves

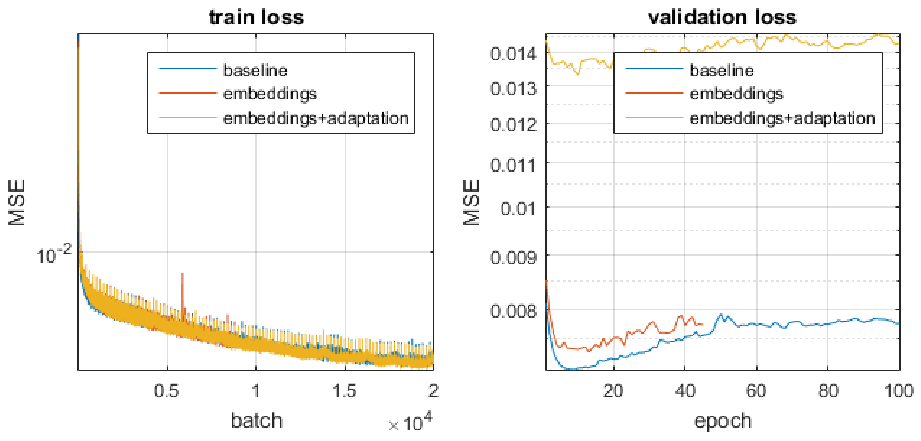


Figure 4.5: Acoustic model's training curves

4.2.2 Subjective evaluation

To perform the subjective evaluation, six sentences were chosen from the test split from the Blizzard corpus. To choose the sentences, all the transcripts from the corpus were processed with the sentiment analysis classifier from Section 3.1 and a score was computed for each of them:

$$s = 2 \cdot p_{nn} + \cdot p_n - \cdot p_p - 2 \cdot p_{pp} \tag{4.1}$$

Where p_{nn} is the probability of the sentence being *very negative*, p_n is the probability of being *negative*, p_p is the probability of the sentence being *positive* and p_{pp} is the probability of being *very positive*. The probability of a sentence being *neutral* is not used in the score.

The scored sentences were sorted in a list and two were picked from the top (as negative sentences), two from the bottom (positive sentences) and two from the middle (neutral sentences).

A web based application (Figure 4.7) was developed with the selected audio files obtained from the experiments and volunteers were asked to rate how appropriate the voices were in a scale from one to five, one being *very inappropriate* and five being *very appropriate*. The results were saved in an SQL database in the server side of the application.

The test was performed by a total of nine volunteers who gave a rating to all the samples, therefore we end up having a total of fifty-four subjective evaluations for each of the three experiments. Figure 4.6 contains a box plot of the collected data:

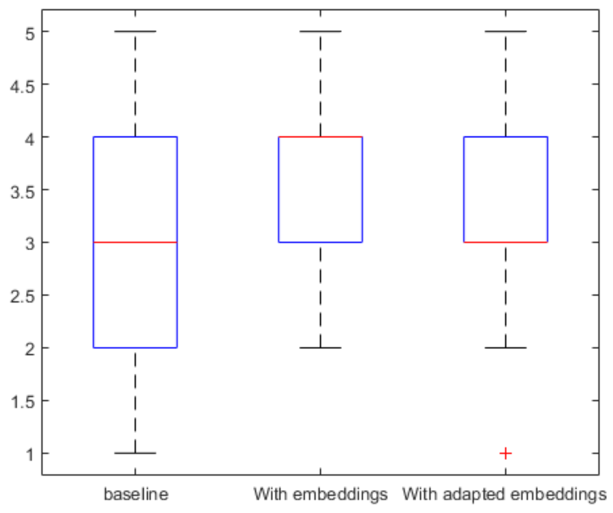
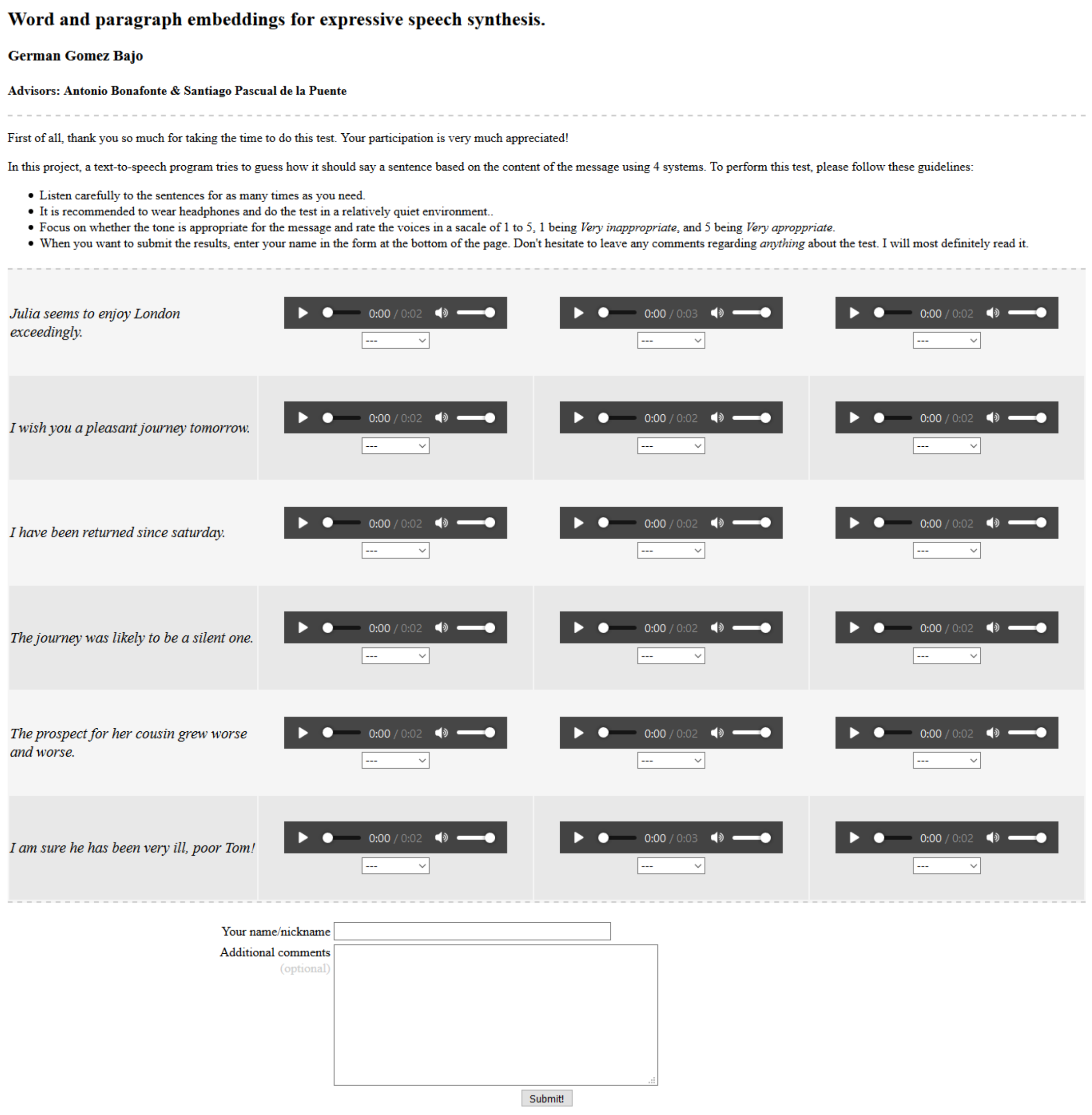


Figure 4.6: Subjective evaluation results.

Which shows that the expressive experiments have had a more positive reception than the baseline.



Word and paragraph embeddings for expressive speech synthesis.

German Gomez Bajo

Advisors: Antonio Bonafonte & Santiago Pascual de la Puente

First of all, thank you so much for taking the time to do this test. Your participation is very much appreciated!

In this project, a text-to-speech program tries to guess how it should say a sentence based on the content of the message using 4 systems. To perform this test, please follow these guidelines:

- Listen carefully to the sentences for as many times as you need.
- It is recommended to wear headphones and do the test in a relatively quiet environment..
- Focus on whether the tone is appropriate for the message and rate the voices in a sacale of 1 to 5, 1 being *Very inappropriate*, and 5 being *Very aproppriate*.
- When you want to submit the results, enter your name in the form at the bottom of the page. Don't hesitate to leave any comments regarding *anything* about the test. I will most definitely read it.

Julia seems to enjoy London exceedingly.	0:00 / 0:02 ---	0:00 / 0:03 ---	0:00 / 0:02 ---
I wish you a pleasant journey tomorrow.	0:00 / 0:02 ---	0:00 / 0:02 ---	0:00 / 0:02 ---
I have been returned since saturday.	0:00 / 0:02 ---	0:00 / 0:02 ---	0:00 / 0:02 ---
The journey was likely to be a silent one.	0:00 / 0:02 ---	0:00 / 0:02 ---	0:00 / 0:02 ---
The prospect for her cousin grew worse and worse.	0:00 / 0:02 ---	0:00 / 0:02 ---	0:00 / 0:02 ---
I am sure he has been very ill, poor Tom!	0:00 / 0:02 ---	0:00 / 0:03 ---	0:00 / 0:02 ---

Your name/nickname

Additional comments (optional)

Figure 4.7: Screenshot of the subjective evaluation's web application.

Chapter 5

Budget

The cost of this project comes from the sever uptime and the salary of the author and advisors.

The project was developed using the computing resources from the VEU lab, and we've estimated its cost by using the pricing from Amazon Web Services.

The supervisors and advisors are assumed to have the wage of senior engineers, and my position to be that of a junior engineer.

The time spent on this project was 20 weeks in total.

	Amount	**Wage/hour**	**Dedication**	**Total**
Junior engineer	1	15.00 €/h	40h/week	12,000 €
Senior engineer	2	60.00 €/h	2 h/week	2,400 €
	Amount	**Hours**	**Cost/hour**	**Total**
Servers	1	3,360 h	0.99 €/h	3,326.4 €
			Total	20,126.4 €

Table 5.1: Estimated budget of the project

Chapter 6

Conclusions

In this project, a speech synthesizer has been developed using the Socrates framework and Keras deep learning library. All the work has been developed with the help of members of the VEU research lab at UPC.

The used databases for training the systems were the Stanford sentiment treebank and the Blizzard challenge. The first one was already prepared for easy use but the second one had to be prepared to be used in the developed speech synthesizers. To perform this step, we used the Ogmios and Ramses software from the VEU research lab. We then used sentiment analysis to obtain expressive information about the training and test data.

The total amount of speech data had to be cut down to 30% of the total so this had an impact in the naturalness of the synthesized voice. But we stayed with this conditions because the goal of the project was to improve the expressiveness and not the clarity of the speech. In the end, the speech is still intelligible.

To adapt the sentiment analysis task to the Blizzard corpus, an adaptation step was performed which in the end did not improve the objective metrics of the system. This could be because the followed strategy was not the appropriate one. This is a potential area where this project could potentially be improved in the future.

Another thing to notice is that the Blizzard corpus contained several audiobooks, but we did not mix them in the train, test and validation sets. An improvement would involve using data from all the audiobooks in the train, test and validation data.

A listening test was performed by nine volunteers who rated the synthesized voices from appropriate to not appropriate in a scale. This subjective evaluation shows some preference to the developed systems. We included speech with and without the adaptation step, but the one without it performed better.

Regarding the initial work plan, a few modifications were made to simplify the system. Initially we had planned to perform the adaptation using a new model instead of reusing the one trained with the Stanford dataset. Also the preparation of the Blizzard corpus was delayed a few weeks from the original plan.

The classical and modern techniques of speech synthesis have been reviewed as part of this project, as well as natural language processing techniques for text classification and sentiment analysis. I also could develop this project in a high-performant computational environment, something I normally don't have access to. I have deepen in statistical parametric speech theory and application, as well as natural language processing. As a side result, there is a new properly segmented corpus available to deepen in the topic of expressive speech synthesis in the future, and a web application that could be reused in the future to perform subjective evaluations.

Bibliography

[1] Yusuf Aytar, Carl Vondrick, and Antonio Torralba. Soundnet: Learning sound representations from unlabeled video. In *Advances in Neural Information Processing Systems*, pages 892–900, 2016.

[2] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[4] Phil Blunsom. Hidden markov models. *Lecture notes, August*, 15:18–19, 2004.

[5] Antonio Bonafonte, Pablo D Agüero, Jordi Adell, Javier Pérez, and Asunción Moreno. Ogmios: The upc text-to-speech synthesis system for spoken translation. In *TC-STAR Workshop on Speech-to-Speech Translation*, pages 199–204, 2006.

[6] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[7] Murtaza Bulut, Shrikanth S Narayanan, and Ann K Syrdal. Expressive speech synthesis using a concatenative synthesizer. In *INTERSPEECH*, 2002.

[8] Yves Chauvin and David E Rumelhart. *Backpropagation: theory, architectures, and applications*. Psychology Press, 1995.

[9] Sin-Horng Chen, Shaw-Hwa Hwang, and Yih-Ru Wang. An rnn-based prosodic information synthesizer for mandarin text-to-speech. *IEEE transactions on speech and audio processing*, 6(3):226–239, 1998.

[10] Gobinda G Chowdhury. Natural language processing. *Annual review of information science and technology*, 37(1):51–89, 2003.

[11] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[12] E. Navas D. Erro, I. Sainz and I. Hernaez. “improved hnm-based vocoder for statistical synthesizers. In Proc. of INTER-SPEECH, editor, *Advances in Neural Information Processing Systems 29*, pages 1809–1812. Curran Associates, Inc., 2011.

[13] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[14] Ellen Eide, Andrew Aaron, Raimo Bakis, Wael Hamza, Michael Picheny, and J Pitrelli. A corpus-based approach to expressive speech synthesis. In *Fifth ISCA Workshop on Speech Synthesis*, 2004.

[15] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[16] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Interspeech*, pages 1756–1760, 2013.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] Andrew J Hunt and Alan W Black. Unit selection in a concatenative speech synthesis system using a large speech database. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 373–376. IEEE, 1996.

[19] Igor Jauk, Antonio Bonafonte, and Santiago Pascual. Acoustic feature prediction from semantic features for expressive speech using deep neural networks. In *Signal Processing Conference (EUSIPCO), 2016 24th European*, pages 2320–2324. IEEE, 2016.

[20] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[21] Simon King and Vasilis Karaiskos. The blizzard challenge 2013. 2013.

[22] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[24] José Bernardo Mariño Acebal, Albino Nogueiras Rodríguez, and Antonio Bonafonte Cávez. The demiphone: an efficient subword unit for continuous speech recognition. In *Proceedings of EUROSPEECH'97*, pages 1215–1218. Editors: G. Kokkinakis, N. Fakotakis, E. Dermatas; Editorial: WCL, University of Patras, Grece, 1997.

[25] Mikiko Mashimo, Tomoki Toda, Kiyohiro Shikano, and Nick Campbell. Evaluation of cross-language voice conversion based on gmm and straight. 2001.

[26] Eric Moulines and Francis Charpentier. Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech communication*, 9(5-6):453–467, 1990.

[27] Santiago Pascual de la Puente. Deep learning applied to speech synthesis. Master's thesis, Universitat Politècnica de Catalunya, 2016.

[28] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[30] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. *Artificial Neural Networks–ICANN*

2010, pages 92–101, 2010.

[31] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, Christopher Potts, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642, 2013.

[32] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[33] Hanna M Wallach. Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*, pages 977–984. ACM, 2006.

[34] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[35] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[36] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[37] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[38] Heiga Zen, Takashi Nose, Junichi Yamagishi, Shinji Sako, Takashi Masuko, Alan W Black, and Keiichi Tokuda. The hmm-based speech synthesis system (hts) version 2.0. In *SSW*, pages 294–299, 2007.

[39] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

Appendix A

Keras model summaries

A.1 Keras summary of the sentiment analysis network

Result of calling the summary method on the keras model.

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 128)           0
____________________________________________________________________________________________________
input_2 (InputLayer)             (None, 128)           0
____________________________________________________________________________________________________
embedding_1 (Embedding)          (None, 128, 300)      120000300   input_1[0][0]
____________________________________________________________________________________________________
embedding_2 (Embedding)          (None, 128, 300)      0           input_2[0][0]
____________________________________________________________________________________________________
convolution1d_1 (Convolution1D)  (None, 126, 82)       73882       embedding_1[0][0]
____________________________________________________________________________________________________
convolution1d_2 (Convolution1D)  (None, 126, 82)       73882       embedding_2[0][0]
____________________________________________________________________________________________________
convolution1d_3 (Convolution1D)  (None, 125, 82)       98482       embedding_1[0][0]
____________________________________________________________________________________________________
convolution1d_4 (Convolution1D)  (None, 125, 82)       98482       embedding_2[0][0]
____________________________________________________________________________________________________
convolution1d_5 (Convolution1D)  (None, 124, 82)       123082      embedding_1[0][0]
____________________________________________________________________________________________________
convolution1d_6 (Convolution1D)  (None, 124, 82)       123082      embedding_2[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 126, 82)       0           convolution1d_1[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)        (None, 126, 82)       0           convolution1d_2[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)        (None, 125, 82)       0           convolution1d_3[0][0]
____________________________________________________________________________________________________
activation_4 (Activation)        (None, 125, 82)       0           convolution1d_4[0][0]
____________________________________________________________________________________________________
activation_5 (Activation)        (None, 124, 82)       0           convolution1d_5[0][0]
____________________________________________________________________________________________________
activation_6 (Activation)        (None, 124, 82)       0           convolution1d_6[0][0]
____________________________________________________________________________________________________
merge_1 (Merge)                  (None, 126, 164)      0           activation_1[0][0]
                                                                   activation_2[0][0]
____________________________________________________________________________________________________
merge_2 (Merge)                  (None, 125, 164)      0           activation_3[0][0]
                                                                   activation_4[0][0]
____________________________________________________________________________________________________
merge_3 (Merge)                  (None, 124, 164)      0           activation_5[0][0]
                                                                   activation_6[0][0]
____________________________________________________________________________________________________
globalmaxpooling1d_1 (GlobalMaxPo(None, 164)           0           merge_1[0][0]
____________________________________________________________________________________________________
globalmaxpooling1d_2 (GlobalMaxPo(None, 164)           0           merge_2[0][0]
____________________________________________________________________________________________________
globalmaxpooling1d_3 (GlobalMaxPo(None, 164)           0           merge_3[0][0]
____________________________________________________________________________________________________
merge_4 (Merge)                  (None, 492)           0           globalmaxpooling1d_1[0][0]
                                                                   globalmaxpooling1d_2[0][0]
                                                                   globalmaxpooling1d_3[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 492)           0           merge_4[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 5)             2465        dropout_1[0][0]
____________________________________________________________________________________________________
activation_7 (Activation)        (None, 5)             0           dense_1[0][0]
====================================================================================================
Total params: 120593657
____________________________________________________________________________________________________
```

A.2 Keras summary of the network used for adaptation

Result of calling the summary method on the keras model.

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 160000, 1)     0
____________________________________________________________________________________________________
convolution1d_1 (Convolution1D)  (None, 80000, 32)     2080        input_1[0][0]
____________________________________________________________________________________________________
batchnormalization_1 (BatchNormal(None, 80000, 32)     64          convolution1d_1[0][0]
____________________________________________________________________________________________________
prelu_1 (PReLU)                  (None, 80000, 32)     2560000     batchnormalization_1[0][0]
____________________________________________________________________________________________________
maxpooling1d_1 (MaxPooling1D)    (None, 10000, 32)     0           prelu_1[0][0]
____________________________________________________________________________________________________
convolution1d_2 (Convolution1D)  (None, 5000, 64)      65600       maxpooling1d_1[0][0]
____________________________________________________________________________________________________
batchnormalization_2 (BatchNormal(None, 5000, 64)      128         convolution1d_2[0][0]
____________________________________________________________________________________________________
prelu_2 (PReLU)                  (None, 5000, 64)      320000      batchnormalization_2[0][0]
____________________________________________________________________________________________________
maxpooling1d_2 (MaxPooling1D)    (None, 625, 64)       0           prelu_2[0][0]
____________________________________________________________________________________________________
convolution1d_3 (Convolution1D)  (None, 157, 128)      131200      maxpooling1d_2[0][0]
____________________________________________________________________________________________________
batchnormalization_3 (BatchNormal(None, 157, 128)      256         convolution1d_3[0][0]
____________________________________________________________________________________________________
prelu_3 (PReLU)                  (None, 157, 128)      20096       batchnormalization_3[0][0]
____________________________________________________________________________________________________
maxpooling1d_3 (MaxPooling1D)    (None, 19, 128)       0           prelu_3[0][0]
____________________________________________________________________________________________________
convolution1d_4 (Convolution1D)  (None, 5, 256)        262400      maxpooling1d_3[0][0]
____________________________________________________________________________________________________
batchnormalization_4 (BatchNormal(None, 5, 256)        512         convolution1d_4[0][0]
____________________________________________________________________________________________________
prelu_4 (PReLU)                  (None, 5, 256)        1280        batchnormalization_4[0][0]
____________________________________________________________________________________________________
convolution1d_5 (Convolution1D)  (None, 5, 512)        524800      prelu_4[0][0]
____________________________________________________________________________________________________
batchnormalization_5 (BatchNormal(None, 5, 512)        1024        convolution1d_5[0][0]
____________________________________________________________________________________________________
prelu_5 (PReLU)                  (None, 5, 512)        2560        batchnormalization_5[0][0]
____________________________________________________________________________________________________
convolution1d_6 (Convolution1D)  (None, 1, 5)          12805       prelu_5[0][0]
____________________________________________________________________________________________________
batchnormalization_6 (BatchNormal(None, 1, 5)          10          convolution1d_6[0][0]
____________________________________________________________________________________________________
prelu_6 (PReLU)                  (None, 1, 5)          5           batchnormalization_6[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 5)             0           prelu_6[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 5)             0           flatten_1[0][0]
====================================================================================================
Total params: 3904820
____________________________________________________________________________________________________
```