Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Paragraph and word embeddings for Expressive Speech Synthesis

Degree's Thesis
Audiovisual Systems Engineering

Author: German Gomez Bajo
Advisors: Antonio Bonafonte and Santiago Pascual de la Puente

Universitat Politècnica de Catalunya (UPC)
2016 - 2017

Abstract

In this project we attempt to enhance the features of a statistical parametric speech synthesis (SPSS) system based on long short-term memory recurrent neural networks (RNN-LSTM) in order to be able to perform better with an expressive corpus.

The corpus that was used came from the Blizzard challenge [21] and contains roughly 20 hours of audiobooks along with the transcripts. In order to obtain the expressive features (embeddings) we train a convolutional neural network (CNN) to predict how good or bad a sentence is using text from the Stanford Sentiment Treebank [27] dataset, and transfer its mapping function afterwards to a second CNN that works with the waveforms of the Blizzard corpus.

We perform an additional adaptation task to the network that works with text in order to adapt it to the Blizzard corpus, because the two datasets contain different semantic content.

Various experiments have been carried out to find the set of expressive features that give the best objective and subjective results (including a baseline) by means of an evaluation done by TODO volunteers.

TODO results

Acknowledgements

Revision history and approval record

Revision	Date	Purpose
0	00/00/2017	Document creation
1	00/00/2017	Document revision
3	00/00/2017	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	00/00/2017	**Date**	00/00/2017	**Date**	00/00/2017
Name	German Gomez	**Name**	Antonio Bonafonte	**Name**	Santiago Pascual de la Puente
Position	Project Author	**Position**	Project Supervisor	**Position**	Project Supervisor

Contents

1 State of the art **8**

1.1 Speech synthesis . . . 8

1.1.1 Unit selection . . . 8

1.1.2 Statistical Parametric Speech Synthesis . . . 10

1.1.3 Recurrent Neural Network-based Speech Synthesis . . . 10

1.1.4 Expressive speech synthesis . . . 12

1.2 Text classification for sentiment Analysis . . . 13

1.3 Deep Learning . . . 13

1.3.1 Deep Neural Networks . . . 14

1.3.2 Optimization . . . 15

1.4 Recurrent networks . . . 16

1.4.1 Long Short-Term Memory (LSTM) . . . 17

1.5 Convolutional Neural Networks . . . 17

2 Expressive Speech synthesis **19**

2.1 Baseline development . . . 19

2.1.1 Data preparation . . . 20

2.1.2 Obtaining Acoustic features . . . 20

2.1.3 Acoustic feature normalization . . . 23

2.1.4 Obtaining linguistic features . . . 24

2.1.5 Linguistic features normalization . . . 27

2.1.6 Baseline training . . . 28

2.2 Expressive synthesis development . . . 30

2.2.1 Sentiment analysis & embedding extraction task . . . 30

2.2.2 Network domain adaptation . . . 32

2.2.3 Training embeddings extraction and expressive synthesis . . . 33

2.2.4 Training the expressive speech synthesizer . . . 34

2.3 Experiments . . . 35

3 Evaluation & Results **36**

3.1 Objective evaluation 36
3.2 Subjective evaluation 36

4 Budget **37**

5 Conclusions **38**

6 Annex **39**

6.1 SA keras summary 39
6.2 Audio keras summary 40

List of Figures

1.1 A 3-state left-to-right Hidden Markov Model. 10
1.2 SPSS system based on HMMs. This scheme corresponds to the HTS framework [34]. 11
1.3 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 [8] . 12
1.4 Basic neuron (left) and neural network configuration (right). 14
1.5 Unfolded RNN 17

2.1 Baseline models. 19
2.2 Full utilization of system resources by Ahocoder. Memory and CPU usage are close to their limits. 22
2.3 F_0 contour interpolation 24
2.4 Training table used to train the acoustic model is stored in the server disk. . 24
2.5 Label representation of the phrase "the autobiography of a horse." 26
2.6 Ogmios modules used to predict phoneme duration. 26
2.7 Histogram of the phoneme duration predictions by Ogmios. 28
2.8 Configuration example for the baseline system. Some options have been omitted. 29
2.9 Example of sentence tree corresponding to the sentence "Effective but too-tepid biopic" 31
2.10 Text classification architecture. The figure is originally from [20] 31
2.11 Distribution of sentence lengths in the Stanford sentiment treebank dataset. . 33

List of Tables

2.1 Information about the Blizzard Challenge utterances. 20
2.2 Label format. 26
2.3 CNN used to process the Blizzard waveforms. 33

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.

Next there is a brief introduction to Deep Learning and deep architectures (both definition and training) a field that is applied to this project. By the end of this chapter, the necessary background will have been introduced for its application in the next chapter.

1.1 Speech synthesis

Speech synthesis is the process of generating a synthetic speech signal, emulating that of a real 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 a human would make to utter said text, and it has to be possible to generate any text that is given to the system.

Speech synthesis has many applications, from medical to recreational. These systems can be used to help speech impaired people and it can also be used to design more human-like interactions objects.

1.1.1 Unit selection

Unit selection systems render a speech signal by concatenating waveform fragments from a large single-speaker database [17]. The outcome of this technique is highly natural-sounding speech since the fragments that are concatenated come from real waveforms. However, while this system is good at producing natural sounding speech, it lacks the flexibility to produce

new speaker sounds or vary the speaking style, as described in [19], since we are limited by the waveforms database. Each unit also has an associated feature vector containing prosodic information such as pitch and formants.

To choose the best sequence of concatenated units, a cost of concatenation is defined [?] given a sequence of units $u_1^N = \{u_1, u_2, ..., u_N\}$ and their corresponding prosodic vectors p_1^N:

$$C_T^p = C_T^p(u_1^N) = \sum_{i=1}^{N} C_p(p_t, p_i) \tag{1.1}$$

$$C_T^c = C_T^c(u_1^N) = \sum_{i=2}^{N} C_c(u_{i-1}, u_i) \tag{1.2}$$

Where C_p is a function that measures the cost of using a specific feature vector (for example if the units have matching formants and pitch) to match a given target p_t and C_c is the cost of concatenation of two consecutive units (for example a unit ends with the same pitch level that the next one starts with). Finding the best sequence of units is done by using the Viterbi algorithm to minimize the total concatenation cost:

$$u_1^N = \underset{u_1^N}{argmax} \left\{ \rho \cdot C_T^p(u_1^N) + (1 - \rho) \cdot C_T^c(u_1^N) \right\} \tag{1.3}$$

$$0 \leq \rho \leq 1 \tag{1.4}$$

Where ρ is a weighting term that can be used to pay more attention to the targets that to the concatenation, or vice versa.

1.1.2 Statistical Parametric Speech Synthesis

In statistical-parametric speech synthesis (SPSS) a set of spectral and excitation parameters are extracted from a speech corpus from where a set of statistical generative models (typically a hidden Markov model or HMM) are trained to model the them for use in speech synthesis.

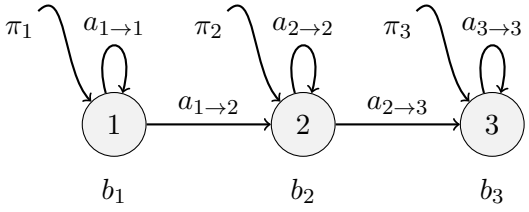

Figure 1.1: A 3-state left-to-right Hidden Markov Model.

As seen in Figure 1.1, a HMM is a finite state machine that can be described by the tuple $\lambda = \langle A, B, \Pi \rangle$ where A are the state transition probabilities, B are the output distributions on each state (spectral & excitation output parameters) and Π are the initial state probabilities. HMMs provide a time-discrete series of observations from a distribution of the speech parameters. In contrast with unit selection, SPSS models offer a more compact representation than unit selection since they don't require a large database after the models have been trained.

Moreover, SPSS also overcomes the limitation of only being limited to one speaker, since we can easily modify the speaker characteristics and speaking styles by transforming the model parameters appropriately as describes in [34] and [**?**].

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 from SPSS.

RNNs are covered in more detail in a later section of this chapter. As describes by [8], this scheme consists of using neural networks to generate parametric speech in two stages:

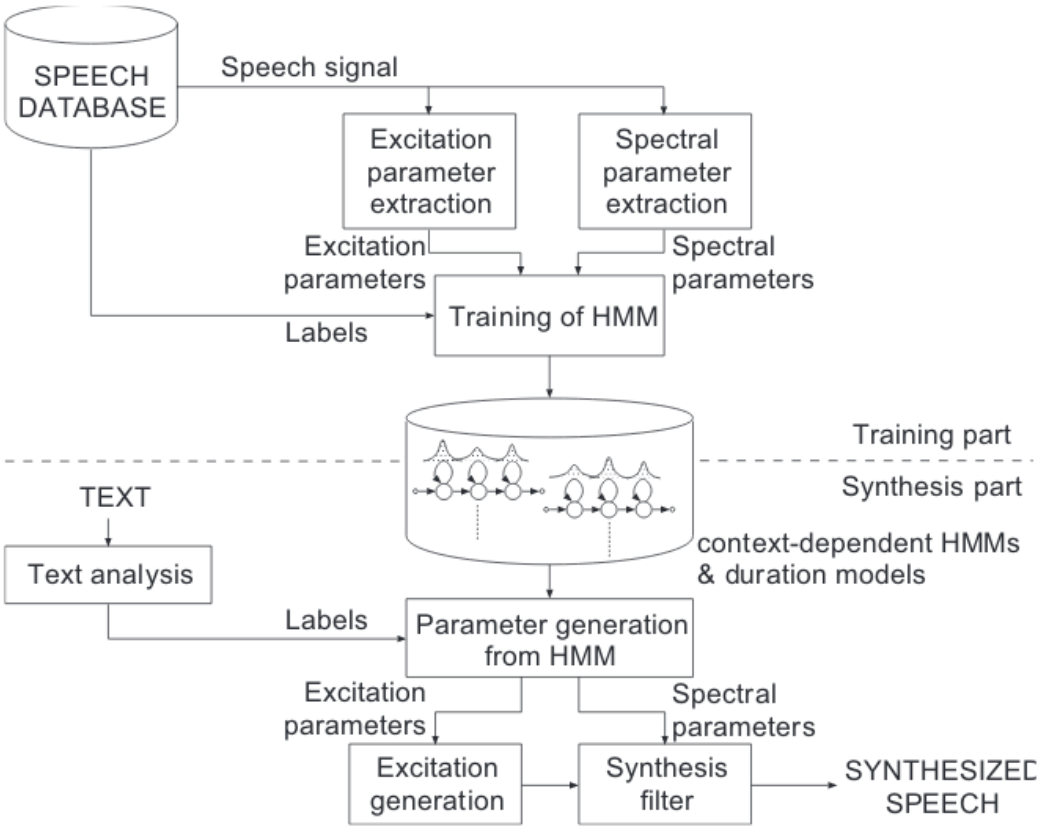

Figure 1.2: SPSS system based on HMMs. This scheme corresponds to the HTS framework [34].

a duration model that predicts the duration of a phoneme in a sentence, and an acoustic model to predict the spectral and excitation parameters for as many time steps as predicted by the duration model. Figure 1.3 shows the complete scheme of this approach.

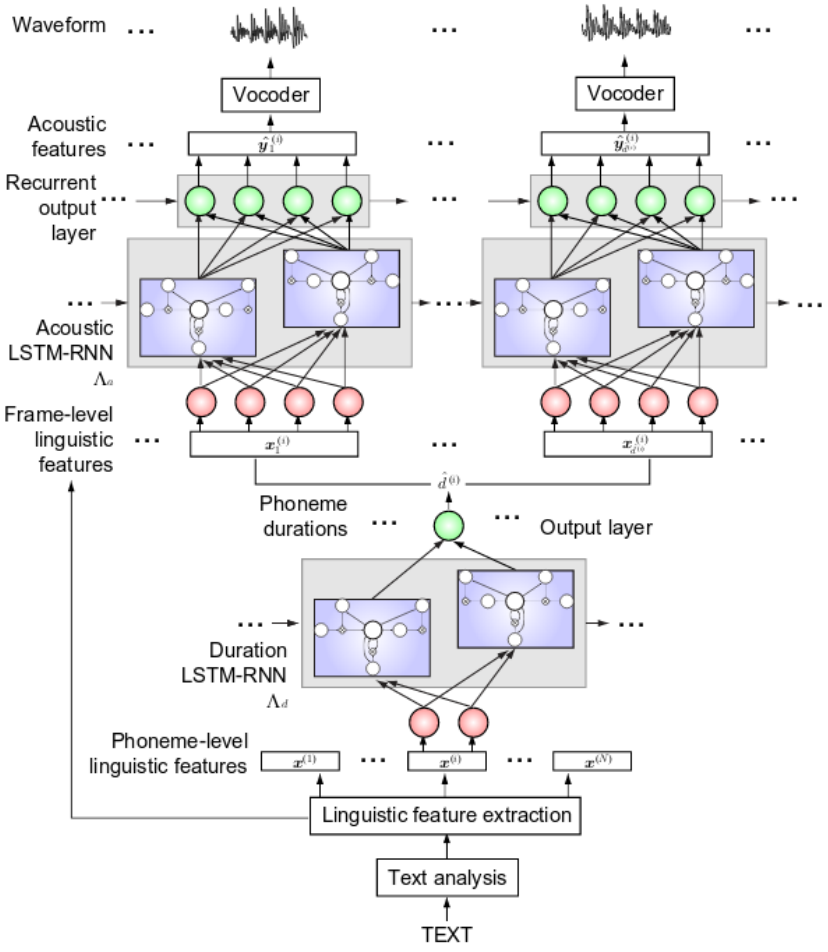

Figure 1.3: RNN-based SPSS scheme. Concepts such as LSTM that appear in the figure are covered at later section of this chapter. The figure is originally from [8]

1.1.4 Expressive speech synthesis

The techniques from the previous sections for doing speech synthesis typically only offer a single speaking style that is independent of the message that is being transmitted. In [6] and [13] a prosodic target is modeled in order to synthesize a text with different types of emotions using a unit selection systems and pitch prediction to modify the prosodic content of the speech.

More recently, [18] uses deep neural networks (more on this in the following sections) to

predict acoustic features of expressive speech from semantic vector space representations. The results can be used to enhance the inputs of speech synthesis systems by modeling the expressiveness of a corpus in SPSS systems, which in contrast with [6] and [13], the synthesis would adapt to the semantic content of the message.

1.2 Text classification for sentiment Analysis

Text classification is a task in natural language processing [9] (NLP) where a text is assigned a given label automatically. A classic model used for text classification is bag of words [29] (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, a SPAM email contains specific words common to all SPAM emails, or a positive text may contain words that are different from a negative text).

More recently, more modern techniques such as deep learning have been used to improve the accuracy of these systems. In [35] text is treated as a signal an processed using Convolutional neural networks (more on this in the next sections) and [20] uses vector representations of words (word embeddings [14]) to classify 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. [12]

[2] 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 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 compute power, large datasets that can take advantage of deep models such Imagenet [25] 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.

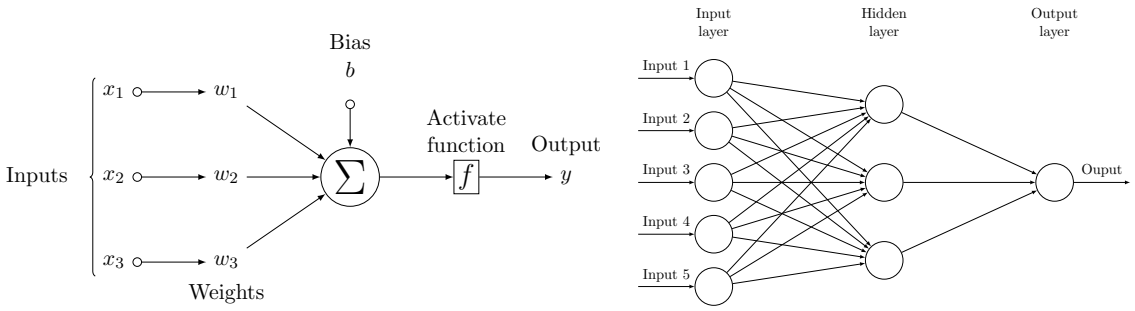

Figure 1.4: Basic neuron (left) and neural network configuration (right).

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 (1.6) or a tanh) which introduces the non-linearities of the system. Equation 1.5 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.

$$\mathbf{y_i} = \sigma\left(\mathbf{W_i} \cdot \mathbf{x_i}\right) \tag{1.5}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.6}$$

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 1.4 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 wether 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 [15]). given a loss, we can improve the performance of a model by translating the weight vector in the direction of the gradient.

$$L(\mathbf{x}; \mathbf{w}) = \sum_{\mathbf{x}_k} L(\mathbf{x_k}) \tag{1.7}$$

$$w_i[n+1] = w_i[n] - \lambda \frac{\partial L(\mathbf{x}; \mathbf{w}[n])}{\partial w_i} \tag{1.8}$$

Equation 1.7 is a loss function, computed by adding the errors of a batch of vectors of a dataset. These vectors are the examples from where the network learns. The size of the batch is usually fixed and smaller than the total size of the dataset, and is used to perform

an iteration in the weight update process (Equation 1.8). This technique is called stochastic gradient descent [5] (SGD).

Before updating the weights, the partial derivatives of the loss function are computed using the back-propagation algorithm [7]. For this we need to be able to compute the derivatives of both the loss function and the activation functions of the neurons, and the update is performed like in Equation 1.8.

The norm of the gradient can be scaled by a factor called learning rate (expressed as λ in Equation 1.8) to speed-up or slow-down the convergence of the loss function during training. Ways to improve this weight updates are Adam [22] and Adadelta [33].

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:

$$\mathbf{y_t} = \sigma\left(\mathbf{W} \cdot \mathbf{x_t} + \mathbf{U} \cdot \mathbf{y_{t-1}}\right) \tag{1.9}$$

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, translate a sequence of words to another a different language, predict the duration of each of the phonemes in a sequence, etc...).

Recurrent networks can still be trained efficiently by using back-propagation through time [30] which is specific case of back-propagation where the errors are also back-propagation back in time. If we expand 1.9 we get:

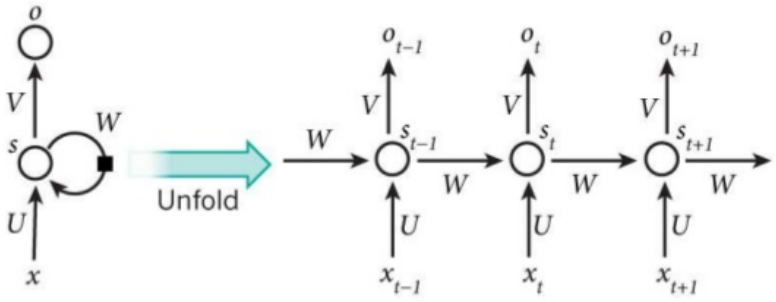

Figure 1.5: Unfolded RNN

$$\mathbf{y_t} = \sigma\left(\mathbf{W} \cdot \mathbf{x_t} + \mathbf{U} \cdot \sigma\left(\mathbf{W} \cdot \mathbf{x_{t-1}} + \mathbf{U} \cdot \sigma\left(\cdots \sigma\left(\cdots\right) \cdots\right)\right)\right) \tag{1.10}$$

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 due to the vanishing or exploding gradient problem [3] that occurs in SGD.

To mitigate this problem in RNNs, we can use specially designed recurrent neurons such as Long Short-Term Memory as opposed to regular RNNs.

1.4.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory [16] or LSTM are a special kind of recurrent unit that deals with the aforementioned problem of the vanishing gradient. This cell works by introducing gating mechanism operated by soft switches trained while optimizing the network that control the flow of information coming in and out of the cell.

1.5 Convolutional Neural Networks

As seen in Equation 1.5, a neuron activation depends on all the output activation of the previous layer. Convolutional neural networks (CNN) are a type of architecture that contain 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 [35].

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.

Chapter 2

Expressive Speech synthesis

This chapter explains how the project was developed. The methodologies and technologies that were used during this project are also discussed.

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 subsequent experiments explained in later sections of this chapter. The architecture is based on the work done by [**?**] by using the Socrates Text-to-speech framework, 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.

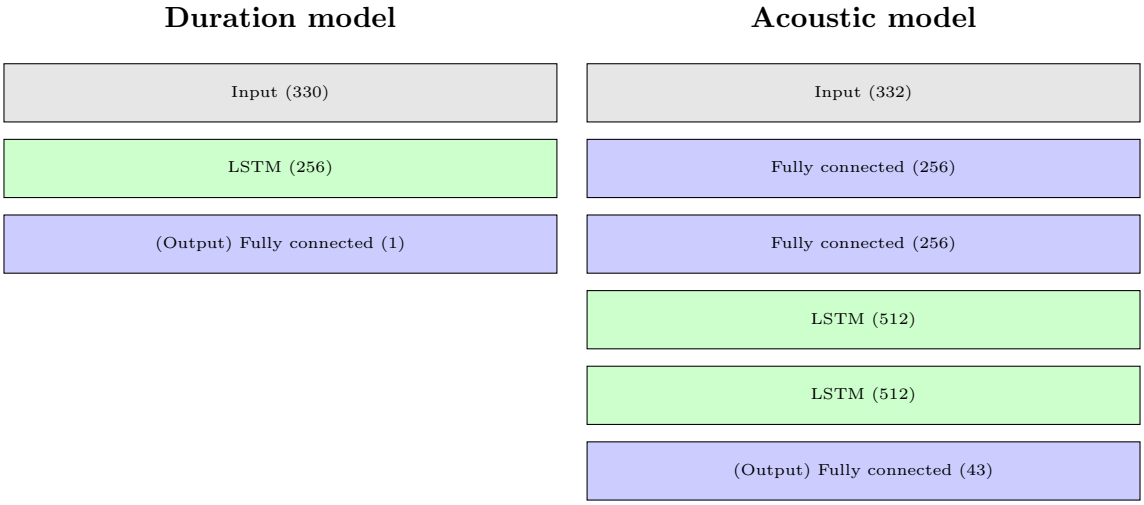

Figure 2.1: Baseline models.

- 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.

2.1.1 Data preparation

The corpus that has been used for this project contains approximately 20 hours or several audiobooks along with the transcripts. The data came from the Blizzard challenge [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.

2.1.2 Obtaining Acoustic features

As mentioned in section 1.1.3, this RNN based speech synthesizer is a SPSS case 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 [11].

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.

- Pitch contour ($log(F_0)$). Unvoiced frames correspond to $F_0 = 0$ value of $-\inf$ after the logarithm. 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 can be obtained using the fv or $log(f0)$ outputs of Ahocoder. This flag is not directly output by the 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 decoder 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 [24] this whole data preparation process was accomplished in less than one hour fully utilizing the system resources as seen in Figure 2.2.

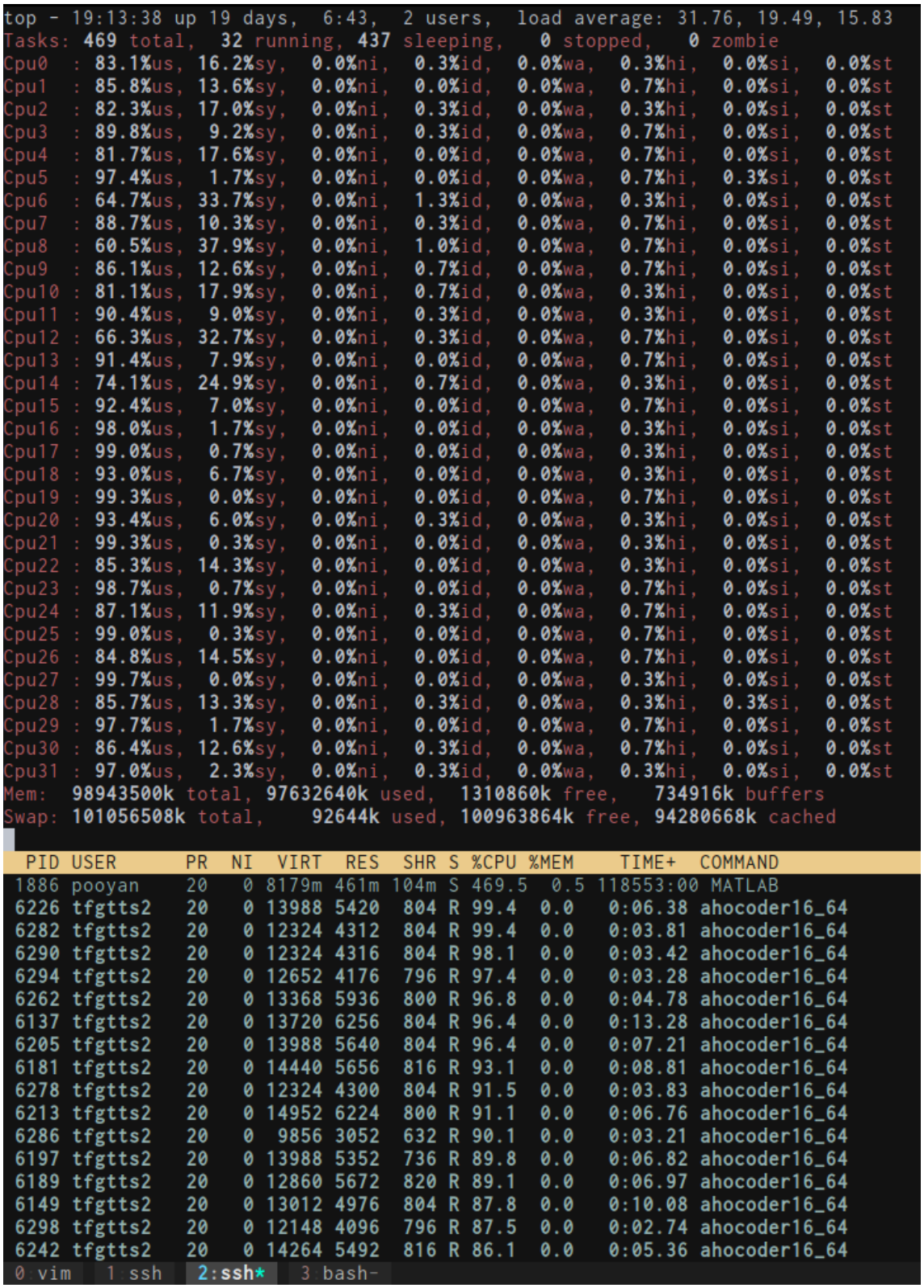

Figure 2.2: Full utilization of system resources by Ahocoder. Memory and CPU usage are close to their limits.

2.1.3 Acoustic feature normalization

The data is not used as it is output by 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 [24].

The outputs of the Ahocoding process ($MFCC$, $log(f_0)$ and f_v) was normalized so that the speech parameters are 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, since their high dynamic range would compress the values from the viced frames too much. [24] 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 2.3 has an example of this interpolation). We can then use the UV flag to recover the original format after denormalization and reconstruct the waveform.

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:

Which are the examples that are used to train the acoustic model. Some of the inputs are

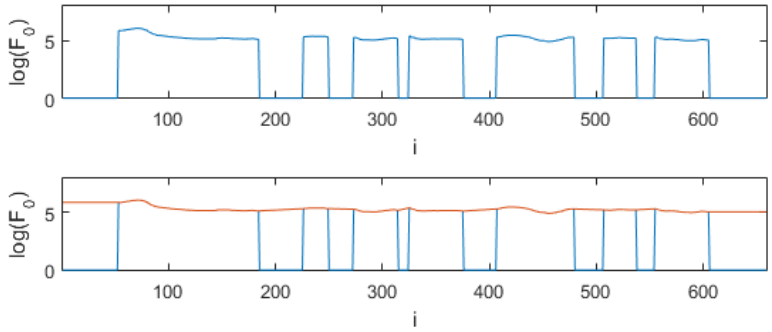

Figure 2.3: F_0 contour interpolation

```
<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 2.4: Training table used to train the acoustic model is stored in the server disk.

repeated depending on the duration of the phonemes. The linguistic features are explained in the next section.

2.1.4 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 **??** 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:

```
p1^p2$-$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)

Table 2.2 (continued)

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

Table 2.2 (continued)

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.

A full example of this conversion is shown in **??**:

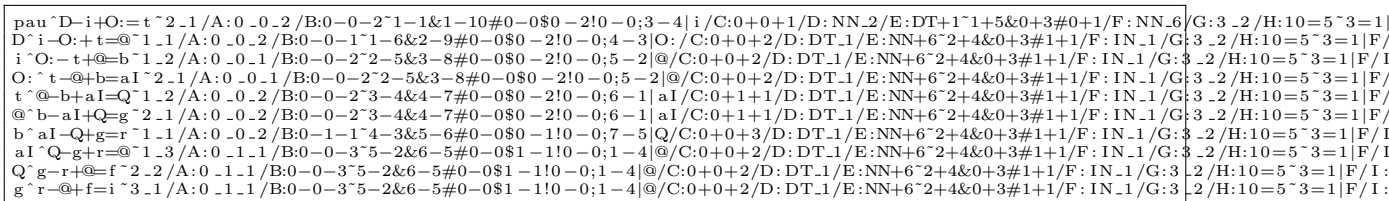

Figure 2.5: Label representation of the phrase "the autobiography of a horse."

This label conversion is carried out with the aid of the Ogmios framework [4]. This program offers a collection of speech processing modules that are combined to perform a given task. For the case of this project, it is used to obtain this label representation and to predict the duration of each of the phonemes in the sentences from the Blizzard corpus. Figure 2.6 shows the steps that it takes to obtain this information:

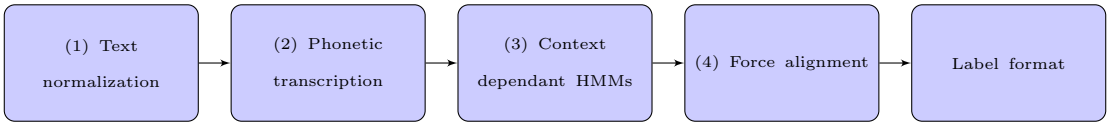

Figure 2.6: Ogmios modules used to predict phoneme duration.

1. The text normalization module converts numeric values, dates, etc to its corresponding

text form.

2. The phonetic transcription module provides the pronunciation of the words.

3. The HMM module estimates a left-to-right HMM model for each of the phonemes (Figure 1.1) using the speech spectral parameters.

4. Given the phonemes of a sentence, it chains the HMM models and the most probable sequence of states is calculates using the Viterbi algorithm. With this information it can calculate the duration of each phoneme by finding the HMM model transitions.

5. The phonetic transcriptions along with the duration predictions are converts to the label format from table **??**.

2.1.5 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 (ME) minimization as discussed in [**?**]. Figure 2.7 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:

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

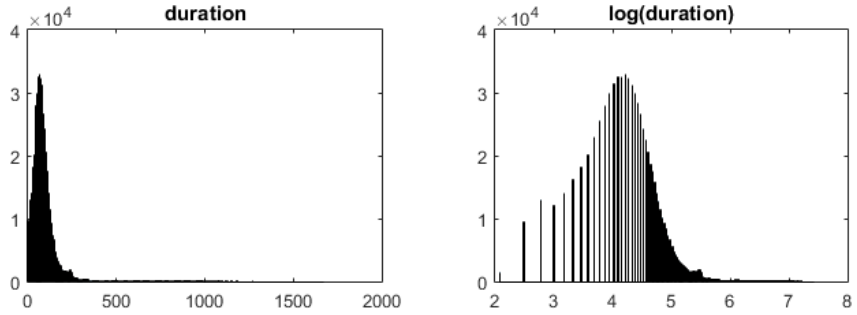

Figure 2.7: Histogram of the phoneme duration predictions by Ogmios.

Once we have obtained the linguistic features, we discard of those who reference the past and only keep the ones tagged with an asterisk ($*$). This is because the models are based on LSTM-RNN which are known to be good at modeling long term dependencies[16].

2.1.6 Baseline training

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 ??.

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 2.1.4 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-

```
[[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.test"
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.test"

[[tts_core]]
name="aco_rnn_core"
dur_stats="model/73/aco/dur.stats.test"
aco_stats="model/73/aco/aco.stats.test"
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 2.8: Configuration example for the baseline system. Some options have been omitted.

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 2.1.2.

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.

6. **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 dataset 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.

2.2 Expressive synthesis development

The baseline system has been explained. 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. 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.

2.2.1 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 task, they lack the information to train a text classification system. To perform this text classification task, we used the Stanford sentiment treebank dataset [27]. An example of a tagged sentence from this dataset is shown in figure **??**.

Each of the nodes on the tree is tagged in a scale from 0 to 4, 0 meaning very negative

```
(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 2.9: Example of sentence tree corresponding to the sentence "Effective but too-tepid biopic"

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 ??.

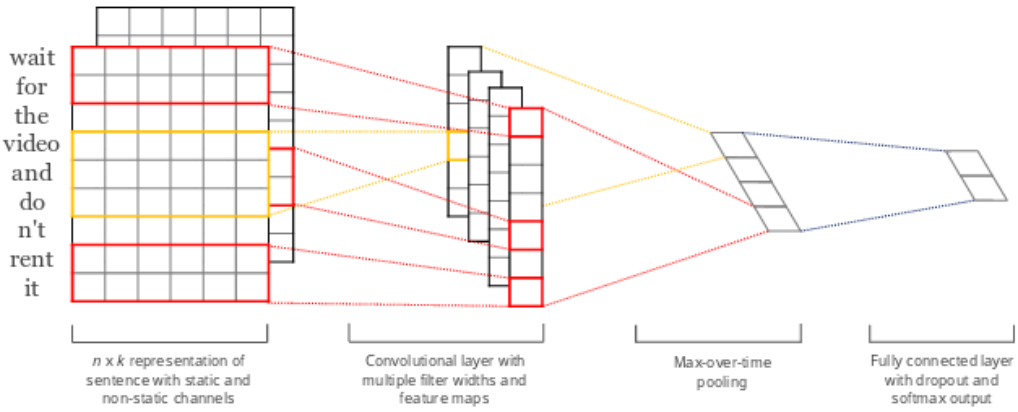

Figure 2.10: Text classification architecture. The figure is originally from [20]

- 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*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.

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

- A max-over-time pooling operation [10], 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 [28] and the weights of this network are constrained using l2 regularization.

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

2.2.2 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. 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 [32] the text network by using labels obtained from the audio network. 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.

2.2.3 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 ??, 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 2.11 shows the distribution of sentence lengths:

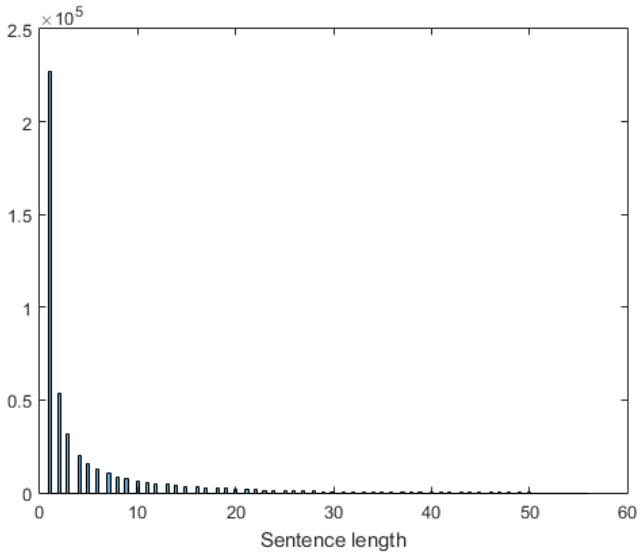

Figure 2.11: Distribution of sentence lengths in the Stanford sentiment treebank dataset.

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.

Domain adaptation task

	conv	pool	conv	pool	conv	pool	conv	conf	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 2.3: CNN used to process the Blizzard waveforms.

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 2.3. 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 [26] and PReLU activations [31] 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.

2.2.4 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.

2.3 Experiments

In order to test the performance of the model, we have performed a series of experiments involving different input configurations of the speech synthesizer models. This experiments are the ones that are shown in the next chapter of this document.

1. Baseline system (only linguistic features from 2.1.4)
2. Text embeddings taken from the sentiment analysis network without doing adaptation step.
3. Text embeddings but with the adaptation step.
4. Train the system using only word embeddings from Glove.

Chapter 3

Evaluation & Results

This chapter shows the results from the different experiments and attempts to fraw conclusions from them.

3.1 Objective evaluation

Training curves of the different stages of this project (accuracy & loss curves) as well as objective metrics from the work proposal document.

3.2 Subjective evaluation

Results from the subjective evaluation. Box plots comparing experiments.

Chapter 4

Budget

An approxmation of the budget of this project, taking into account server uptime, weekly mettings, etc..

Chapter 5

Conclusions

Chapter 6

Annex

6.1 SA keras summary

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

6.2 Audio keras summary

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			

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] 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.

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

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

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

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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.

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

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

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

[17] 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.

[18] 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.

[19] Igor Jauk, Antonio Bonafonte Cávez, Paula López Otero, and Laura Docio Fernández. Creating expressive synthetic voices by unsupervised clustering of audiobooks. In *INTERSPEECH 2015: 16th Annual Conference of the International Speech Communication Association: Dresden, Germany: September 6-10, 2015*, pages 3380–3384. International Speech Communication Association (ISCA), 2015.

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

[21] Simon King and Vasilis Karaiskos. The blizzard challenge 2016. 2016.

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

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

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

[32] 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.

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

[34] 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.

[35] 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.