BUILDING A NATURAL SOUNDING TEXT-TO-SPEECH SYSTEM FOR THE NEPALI LANGUAGE: RESEARCH AND DEVELOPMENT CHALLENGES AND SOLUTIONS

Roop Shree Ratna Bajracharya, Santosh Regmi, Bal Krishna Bal, Balaram Prasain

Text-to-Speech (TTS) synthesis has come far from its primitive synthetic monotone voices to more natural and intelligible sounding voices. One of the direct applications of a natural sounding TTS systems is the screen reader applications for the visually impaired and the blind community. The Festival Speech Synthesis System uses a concatenative speech synthesis method together with the unit selection process to generate a natural sounding voice. This work primarily gives an account of the efforts put towards developing a Natural sounding TTS system for Nepali using the Festival system. We also shed light on the issues faced and the solutions derived which can be quite overlapping across other similar under-resourced languages in the region.

Keywords: Nepali Text-To-Speech, Festival Speech Synthesis, Unit Selection Speech Synthesis

1. Introduction

Text-To-Speech (TTS) system converts the given text to a spoken waveform through text processing and speech generation processes. These processes are connected to linguistic theory, models of speech production, and acoustic-phonetic characterization of language (Klatt 1987). The text to speech synthesis process basically consists of two modules: a Natural Language Processing (NLP) module that produces a phonetic transcription of the input text with desired prosodic features and a Digital Signal Processing (DSP) module that transforms the symbolic information received from the NLP module to speech (Dutoit 1999).

There are three approaches to TTS system: 1) articulatory-model 2) parameter-based and 3) concatenation of stored speech (Klatt 1987). This project deals with the third approach to concatenate stored speech segments (units).

1.1 Concatenative speech synthesis

A concatenative speech synthesis system produces sound by combining recorded sound clips. The recorded speech can be further broken down into smaller components, so that it can be used to make any spoken word. There are three main types of concatenative synthesis (Kishor et al. 2002):

1. Unit selection synthesis that uses large databases of recorded speech and creates database from recorded utterance.
2. Diphone synthesis that uses a minimal speech database containing all the diphones (sound-to-sound transitions) occurring in a language.
3. Domain-specific synthesis which concatenates prerecorded words and phrases to create complete utterances. This approach is useful for limited domain applications like talking clocks, station announcement systems which can be implemented by using only a small number of unique recorded speech.

We have used the unit selection method for this project as it produces the most natural speech output.

1.2 Unit selection speech synthesis

A unit selection model for speech synthesis heavily depends upon a well-organized unit database. The database contains the speech units from a large corpus that includes variety of phonetic and prosodic variants of each unit (Dong et al. 2008). Each instance of a unit in the speech database is described by a vector of features which may be a discrete or continuous value. These features of the unit are used for selecting the correct unit that meets the segmental requirement. Various factors like distance between the selected unit and the desired unit, degree of continuity between the selected units are checked before selecting the units of speech (Dong et al. 2008).

The best unit that phonetically and prosodically match the target units is selected based upon:
1. The appropriateness of the candidate unit compared with target unit.
2. The smoothness between the selected units to be connected.

1.3 Architecture of Nepali TTS engine

The voice synthesis architecture includes mainly three phases: data preparation phase, model building phase, and testing phase. Figure 1 shows the major components of this process (Kishore et al. 2002).

After the voice has been synthesized, we port it to Non-Visual Desktop Access (NVDA) screen reading software using the Festival Add-on available for NVDA.

1.4 Other existing Nepali TTS systems

Currently the only other Nepali TTS system being used in NVDA is eSpeak which uses formant synthesis method and has a small memory footprint. But it is not as natural and smooth as the speech synthesizers based on human speech recordings. Apart from this, there has been a number of research and study conducted in the field of Nepali TTS using concatenative speech synthesis approach (Ghimire and Bal, 2017; Chettri and Sah, 2013).

2. Voice building process

2.1 Data creation and preparation

2.1.1 Corpus building

The data was preliminarily selected as a subset of the Nepali National Corpus (NNC) developed by the Bhasha Sanchar project and later on owned by the Language Technology Kendra. Around 5000 sentences with maximum of 10 words per sentence were selected and analyzed for maximum coverage of all the Nepali phones. Since the randomly selected sentences did not cover the entire diphone coverage, 394 manually constructed sentences were added to fill the missing phones. Therefore, in total we based our analysis on 5394 sentences for the first phase. For the second phase, with an aim to further increase the size of the database, we obtained the random text from different sources and selected around 4241 more sentences. Each of the selected sentences were manually checked for standard spellings, date, time, number, and then normalized and finally recorded.

2.1.2 Recording of the corpus

The prepared sentences were then recorded in a quiet studio setting with the voice of a professional Nepali speaker. The recordings were recorded in 16 KHz bitrate and in mono channel Black and Lenzo 2014).

2.1.3. Verification of recording

The recorded sentences were further analyzed by manually listening to it for mispronunciations and the data was edited accordingly.

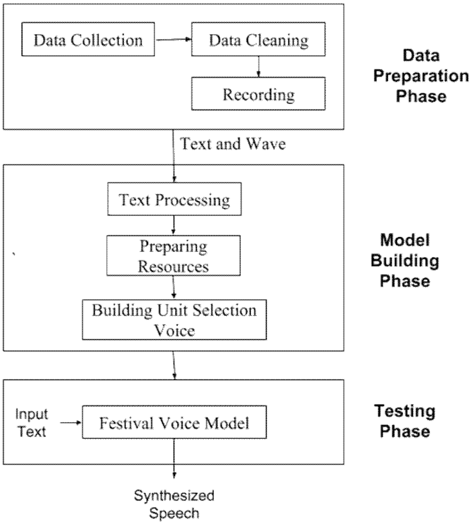

Figure 1: Voice synthesis using Festival and Festvox

2.2 Model building phase

2.2.1 Text processing

The text processing stage includes preprocessing of text to produce the text corpus in the suitable format as required by the Festival System.

2.2.1.1 Transliteration of sentences

Since the Festival System only accepts ASCII code, we transliterated the text from Devanagari Unicode to its corresponding English form. For this purpose, we used a unique mapping for each Nepali phone to convert it into English as shown in Table 1. Transliteration is carried in two levels, first transliteration of Devanagari symbols into

ASCII followed by removing the unnecessary sequences created due to inherent vowel and dependent vowels.

2.2.1.2 Syllabication of words

Syllabication is basically the separation of word into its syllables Treiman and Zukowski. 1990). Before we can build the voice from the given corpus, we need a database which consists of unique words and its syllabified form. For this process, we used a python library that syllabifies the words into its syllables using the sonority theory. This process required us to separate the Nepali letters into following categories:

1. vowels: ax, axn, aa, aan, i, in, u, un, e, en, o, on, axj, axjn, aaj, aajn, ej, ejn, oj, ojn, uj, ujn, aw, awn, aaw, aawn, ew, ewn, ow, own, iw, iwn
2. approximates: y, w
3. nasals: m, n, ng, nn
4. fricatives: s, h
5. affricates: ch, chh, jh, jhh
6. stops: k, kh, g, gh, t, tt, th, d, dd, dh, tth, ddh, p, ph, b, bh, r, l

After the first level of syllabication has been performed by the script, we manually verified each syllabication to ensure accuracy. The syllabified words were converted into proper LISP format required by the Festival System which is saved in the lexicon file.

```
("aabhaasaxmaa" n (((aa)1) ((bh aa s PAU) 0) ((m aa)0)))
```

2.2.2 Preparing resources

For building the voice, we used a number of scripts available in Festvox to setup the voice directory and run the binaries of the Festival Speech Synthesis. The command for creating the folders required for building the voice based on unit selection method in Festival is as follows:

```
$FESTVOXDIR/src/unisel/setupclunits INST LANG VOX
```

Here INST is the institute building the language, e.g. ku for Kathmandu University, LANG is the language parameter e.g. NE, EN etc. and VOX is speaker/style identifier e.g kal, awb. This script creates all the necessary folders and files required for the voice building process.

2.2.2.1 Defining phoneset

Phoneset is a set of smallest discrete segments of sound in a stream of speech. Each language has its own phoneset or alternatively also may share phoneset with other similar languages. To build a complete voice, we need corresponding mapping for each letter in Nepali Devanagari to English as seen in the transliteration process. The phoneset should also include phonetic features for each individual phone. The created phoneset was then placed in the Festvox directory previously created by the above script.

2.2.2.2 Arranging other files

Apart from phoneset, we also arranged the main text data prepared for the recording in its transliterated form. The data was numbered according to the wave recording filename and complied into a single file named txt.done.data. We collected and included 9628 sentences for the full build. The line was arranged in the following format:

(arctic_1101000001 "mulyax saxttthi ddaxlaxrax thiyo ")

The lexicon file was placed in the festvox folder.

Table 1: Transliteration mapping for Nepali phones

Nepali	Trans-literation	Nepali	Trans-literation	Nepali	Trans-literation	Nepali	Trans-literation
आइ	aaj	◌ोऊ	$ow	क	kax	व	wax
◌ाइ	$aaj	◌ेइ	$ej	ख	khax	श	sax
ओइ	oj	औ	aw	ग	gax	ष	sax
◌ोइ	$oj	◌ौ	$aw	ञ	nax	स	sax
ओई	oj	आ	aa	ट	ttax	ह	hax
◌ोई	$oj	◌ा	$aa	ठ	tthax	◌ं	m
एई	ej	इ	i	ड	ddax	◌ँ	n
◌ेई	$ej	◌ि	$i	ढ	ddhax	◌ः	
उइ	uj	ई	i	ण	nnax	◌्	$
◌ुइ	$uj	◌ी	$i	त	tax	इउ	iw
उई	uj	उ	u	थ	thax	◌िउ	$iw
◌ुई	$uj	◌ु	$u	द	dax	इऊ	iw
आउ	aaw	ऊ	u	ध	dhax	◌िऊ	$iw
◌ाउ	$aaw	◌ू	$u	न	nax	आई	aaj
आऊ	aaw	ए	e	प	pax	◌ाई	$aaj
◌ाऊ	$aaw	◌े	$e	फ	phax	एइ	ej
एउ	ew	ओ	o	ब	bax	घ	ghax
◌ेउ	$ew	◌ो	$o	भ	bhax	ङ	Ngax
एऊ	ew	ऋ	ri	म	max	च	Chax
◌ेऊ	$ew	◌ृ	$ri	य	yax	छ	chhax
ओउ	ow	ऐ	axj	र	rax	ज	Jhax
◌ोउ	$ow	◌ै	$axj	ल	lax	झ	jhhax
ओऊ	ow	अ	ax				

2.2.3 Building unit selection voices

Once all the necessary files were made ready, we worked on the build process (Levin and Black 2018).

2.2.3.1 Building the prompts

This step involves generating the prompts for aligning the speech data. This process uses the lexicon files and wave files. All the unique words present in etc/txt.done.data needs to be present in the lexicon file. Then we ran the following script to build the prompts.

```
./bin/do_build build_prompts
```

This built utterances for each sentence and saved waveforms and labels for aligning.

2.2.3.2 Aligning the prompts

The wave files of sentences were first placed in the recording's directory. Then we ran the following script to convert the wave files into suitable format with correct bitrate and channels by power normalizing the wave files.

```
./bin/get_wavs recordings/*.wav
```

Then we labeled each sentence with its corresponding wave files which determines the segment of the wave file has which particular phone.

```
./bin/do_build label
```

Next, we took the phone labels and integrated them into the utterance structure that identifies the words, syllables, and phones which is used to extract information for the voice build (Levin and Black 2018). This is done using the following script:

```
./bin/do_build build_utts
```

2.2.3.3 Speech parameterization

The next step is to analyze the speech. We first found the pitch periods, which determines when the glottis is opened and closed while speaking. Then we found the spectral properties of the speech at each pitch period caused by the vocal tract shape. These step were executed using the following scripts:

```
./bin/make_pm_wave wav/*.wav
./bin/make_pm_fix pm/*.pm
./bin/make_f0_pm wav/*.wav
./bin/make_mcep wav/*.wav
```

We used the following parameters for the pitchmarks settings:

```
FEMALE_ARGS='-min 0.00333 -max 0.0075 -def 0.006 -wave_end -lx_lf 220 -lx_lo 121 -lx_hf 80
-lx_ho 51 -med_o 0'
```

2.2.3.4 Building the cluster unit selection synthesizer

First we ensured that the phonetic features to be included during the voice building process is in synchronization with the features used in the phoneset. This information is included in the festival/clunits/all.desc file. Finally we build the voice using the script:

```
./bin/do_build build_clunits
```

This created a catalogue file that contains information about the location of each unit in the wave file, its duration and its starting and end points along with prediction trees and feature file. The information is arranged in the following format:

r_16286 arctic_1207005454 0.020000 0.100000 0.180000

The prediction trees generated for each phone determines which phone to pick based on the result of the decision tree. The feature file contains the feature information for each unit for each phone.

2.2.4 Additional steps

2.2.4.1 Building letter to sound rules

The generation of voice primarily depends upon the correct pronunciation of the words. But it is not possible to incorporate all the words into the lexicon. Thus to tackle this problem, letter to sound (LTS) rules are used to predict pronunciations for words that are not in the lexicon (Black et al. 1998). Usually the words with the most non-standard pronunciations, should be listed in the lexicon itself. By training the system with a good amount of LTS rules, we can even reduce the size of the lexicon itself.

We trained the LTS rules using the existing lexicon dictionary and the tools available in Festival and FestVox. Given below are the steps to build LTS rules Black and Lenzo, 2014):

1. Prepare the lexicon by removing syllables, part of speech tags and stress marks from the normal lexicon file: *("horax" nil (h o r ax))*

2. Iteratively create allowables.scm file by first including basic mapping for each letter and then aligning each letter.

3. After the allowables.scm has been prepared, we run the following script to generate the LTS rules.

$FESTVOXDIR/src/lts/build_lts

2.2.4.2 Building duration models

Duration models help to predict the multiplication factor for a segment in the input sentence which determines how long the segment is stretched during the utterance of speech. We can build a duration model by using zscores, which is number of standard deviations from the mean (Black and Lenzo 2014). The duration model takes a CART (Classification and Regression Trees) tree that returns zscores by calculating mean and standard deviation from the data. The tree is later converted into absolute duration stretch value and used during the voice synthesis. We can generate the duration model using *make_dur_models* script available in Festival System.

2.3 Testing phase

2.3.1 Running the voice

2.3.1.1 Running voice in Festival in Linux Machine

We tested the voice by first loading the voice's clunit file named *INST_LANG_VOX_clunits.scm.*

```
$festival/bin/festival –b festvox/ku_ne_nab_clunits.scm
```

Then in the Festival command line interface, we sequentially ran the following steps to synthesize the voice and make it speak using the SayText function. Next, we provided the transliterated version of the input text to the Festival System.

```
festival> (voice_ku_ne_nab_clunits)
festival> (SayText "Naxmaxste")
festival> (set! utt1 (SayText " Naxmaxste "))
festival> (utt.save.wave utt1 "out.wav")
```

2.3.1.2 Running the voice in NVDA using Festival add-on

After building the voice, we ported it to the Non-Visual Desktop Access (NVDA) screen reading software using the Festival Add-on available for NVDA. But before we could use our Nepali voice in default Festival Add-On we needed to make sure that the input text gets transliterated accordingly. After this process, the NVDA handled all the input output and provides Festival Engine with the text input, which our voice package synthesized accordingly using the Festival DLL files

2.3.2 Benchmarking the generated speech

We tested a number of test sentences extracted from different Nepali news websites and categorized the output speech into two classes: good and bad based upon the quality of the speech. The percentage of each quality of sentence can be seen on Table 2. We further classified the bad sentences into the type of problem as seen in Table 3.

Table 2: Percentage of generated sentences according to quality

Good Sentences	Bad Sentences
84.21%	15.79%

Table 3: Percentage of classification of problems in sentences

Problems	Percentage
Echoes	23.08%
Overlaps	30.77%
Mispronunciation	7.69%
Echoes/overlaps	38.46%

2.4 Voice synthesis process

When the *SayText* function is called in Festival first the system searches for the corresponding pronunciation for each word separated by space in the lexicon dictionary. The pronunciations for the words not available in the lexicon are generated by the use of

LTS rules. After this process, the system picks an appropriate unit from the speech database for each individual phone using CART generated during the build process (Edinburgh Speech Tools, December 2014). The question nodes use various features and cost function included during the build process to form a decision tree. The values for the features are obtained from the individual phone's feature file.

When all the units have been picked along with their prosodic properties, the corresponding utterance for each unit are picked from their corresponding wave files. The index of each unit and its wave file along with the start and end point of that particular unit are located in the catalogue file. These units are finally concatenated and speech is synthesized through the speech output device or saved to a wave file according to the configurations.

3. Challenges and solutions

3.1 Issue in voice building due to large amount of data

One of the major issues during the voice building process was the huge size of the speech corpus. At the final stage we had a total of 9638 sentences. While building cluster units during the final stage of voice building process, Festival creates distance tables to help in calculating the cost functions that determines the appropriateness of a selected unit. These distance tables require enormous amounts of space depending upon the number of units of a single phone. This process requires huge amounts of RAM in the system during the build process.

To tackle this issue, we arranged the units in the ascending order so that the phone with largest number of units are processed at the last. Then we canceled the distance table building and tree building process for these problematic units and used lesser data to create trees for these selected few phones. This way we could build the voice using the whole corpus.

3.2 Slow response time during speech synthesis

Because we used a large speech corpus in order to get more natural voice, it resulted in the formation of a heavier CART. This caused voice loading time and synthesis time to be relatively higher. (Kumar et al. 2013) suggests using CART pruning to reduce the response time comparatively. We reduced the minimum cluster size to 8 and the maximum number of units that will be in a cluster at a tree leaf to 2 to implement pruning of CART. We received a gain of 0.6 seconds in response time after this implementation. This also reduced the size of our trees by a small amount.

3.3 Other possible voice synthesis approaches

There are many other voice synthesis approaches within Festival voice synthesis environment. But the unit selection method synthesizes the most natural sounding voice among all the other approaches. This is why we chose this approach. Some of the other possible approaches are explained below.

3.3.1 Diphone based speech synthesis

Diphone synthesis is one of the most popular methods used for synthesizing voice from recordings. This method can generate voice faster than the unit selection method and requires less amount of data to train. But, the major defect of this kind of synthesis is that the resulting voice is robotic.

The Diphone method uses two adjacent half-phones from words then, cuts the units at the points of relative stability, rather than at the volatile phone-phone transition (Lenzo and Black, 2000). Thus, with an n-phone inventory there can be (n * n) diphone inventory which creates a synthesizer that can potentially speak anything.

3.3.2 Statistical parameterization based speech synthesis

Voices built from the Festival based unit selection synthesis can produce high quality natural sounding synthetic speech. But these voices are bulky and requires large amounts of space to store all the data and also have inflexible speech tempo. Statistical parametric models for speech acts as an alternative method for speech synthesis to these kinds of voices. This method is more robust to errors and allow for better modeling of variation (Black, 2006). But these voices tend to produce somewhat robotic voices. The CLUSTERGEN synthesizer is implemented within the Festival/FestVox voice building environment using statistical parameterization technique. This method uses an average of a number of units, rather than a set of instances for synthesizing the speech in the unit selection.

4. Conclusion

In this work, various approaches of building a TTS and particularly the Festival system was studied and analyzed in larger detail. We successfully built a Nepali voice that can be used along with NVDA to read the Devanagari text displayed on the screen. However, there are still a few issues of overlaps and echoes in the generated voice which we suspect is due to the potential mislabeling between recorded speech phones and the corresponding characters. We aim to improve on this in future.

Acknowledgements

This work was carried out at the Information and Language Processing Research Lab, Kathmandu University in collaboration with the Nepal Association of Blind with the support of the Kadoorie Charitable Foundation, Hong Kong.

References

Edinburgh Speech Tools, December 2014. http://www.cstr.ed.ac.uk/projects/festival/ [Accessed on 20 June 2018].

Black, Alan W. 2006. CLUSTERGEN: A Statistical Parametric Synthesizer using Trajectory Modeling. *INTERSPEECH 2006 - ICSLP*. [http://festvox.org/bsv/c3174.html].

Black, Alan W. and Kevin Lenzo. 2014. Building Synthetic Voices. Language Technologies Institute, Carnegie Mellon University. [http://festvox.org/bsv/].

Black, Alan W., Kevin Lenzo and Vincent Pagel. 1998. Issues in Building General Letter to Sound Rules. *The Third ESCA Workshop in Speech Synthesis*. [https://pdfs.semanticscholar.org/21d6/baca37dbcb35cf263dd93ce306f6013a1ff5.pdf].

Chettri, Bhusan and Krishna Bikram Shah. 2013. Nepali Text to Speech Synthesis System using ESNOLA Method of Concatenation. *International Journal of Computer Applications*, 62. 24-28. 10.5120/10053-4909.

Dong, Minghui, Kim, Teng Lua and Haizhou Li. 2008. A Unit Selection-based Speech Synthesis Approach. Journal of Chines Language and Computing. 16. 135-144.

Dutoit, Thierry. 1999. A Short Introduction to Text-to-Speech Synthesis. [e-book: http://www.bookmetrix.com/detail/book/d8f34e8a-ac30-4b57-bba2-c282685a5e90].

Ghimire, Rupak Raj and Bal Krishna Bal. 2017. Enhancing the quality of Nepali Text-to-Speech Systems. In Kravets A., Shcherbakov M., Kultsova M., Groumpos P. (eds) *Creativity in Intelligent Technologies and Data Science* [CIT&DS] 2017, vol. 754. 187-197.

Kishore, S. P., Rajeev Sangal and M. Srinivas. 2002. Building Hindi and Telugu Voices using Festvox.[https://www.researchgate.net/publication/242083170_Building_Hindi_and_Telugu_Voices_using_Festvox/link/5625f67708aeedae57db1004].

Klatt, Dennis. H. 1987. Review of text-to-speech conversion for English. *The Journal of the Acoustical Society of America,* 82(3). 737-793.

Kumar, Pranaw, Gandhi Annamalai, Sajini T, Anand Konjengbam, Praveen M and Kasturi R. G. 2013. Seamless Integration of Common Framework Indian language TTSes in various applications. [https://ieeexplore.ieee.org/document/6709873].

Lenzo, Kevin A. and Alan W. Black. 2000. Diphone Collection and Synthesis. ICSLP 2000. [https://www.cs.cmu.edu/~awb/papers/ICSLP2000_diphone/index.html].

Levin, Lori and Alan W. Black. 2018. Building a Talking Clock. http://tts.speech.cs.cmu.edu/11-823/hints/clock.html. [Accessed 20 June 2018].

Treiman, Rebecca and Andrea Zukowski. 1990. Toward an Understanding of English Syllabification. *Journal of Memory and Language,* 26. 66-85.