KTH, Systems, Control and Robotics
Speech and Speaker Recognition (DT2118)

Phonological rules for forced alignment

Submitted by

Vincent Trichon 920513-2633

June 9, 2015

Abstract

We study how phonological rules can be added to the WaveSurfer ASR plugin in order to improve the phonetic match on a forced alignment task. We implement a general system to apply optional rules on each word separately, and test it with a few selected rules against some sentences from the TIMIT corpus.

1 Introduction

In automatic speech recognition, the usual task is, given a recording of some spoken text, to reconstruct that unknown text. Forced alignment is a different task: now both the recording and the transcription are given, and the goal is to align them, that is to find the timestamps at which each of the phonemes in the text are uttered.

What is the point in recognizing a text that is already known ? This is actually important in many applications. In particular, it is used to construct data in order to train recognition models such as models based on neural networks. It is possible to construct train data by hand, but manual alignment is time-consuming and thus costly, and potentially error-prone, so that it is feasible only for relatively small corpora. For a large corpus it is much more efficient to use forced alignment (and possibly hand-check it afterwards). Such a method was used for instance for the TIMIT corpus [1, 5].

Forced alignment seems a much easier task than recognition of an unknown text. Indeed, the search space is largely reduced since we know the transcription. However, some difficulties arise. One of them is that the actual phonemes uttered may differ from the standard prononciation obtained from dictionaries. There are variations among speakers, due to regional variants, and even the same speaker may use different pronunciations for the same word, for instance by dropping or altering some phonemes in a more casual speech. Also, the pronunctiation of a word may be modified by the adjacent words. To take this into account, one possibility would be to store all pronunciation variants of each word in the dictionary, but this requires a lot of work and is quite rigid. A more interesting possibility is to use phonological rules, that express how some phonemes in a certain context can be transformed.

Our goal is to setup a forced alignment procedure that makes use of phonological rules, and to test it.

2 Method

We use the WaveSurfer ASR plugin [3], which in turn uses the Julius ASR engine [8]. This plugin currently has a forced alignment function, but without the use of phonological rules.

We first have to collect a few phonological rules for American English, then to implement a modification of the grammar construction routine in the WaveSurfer ASR plugin to include variations according to those rules.

To test the modified plugin, we need to run it on data that is already aligned so that we can compare its results with the provided alignment.

3 Data

3.1 Phonetic transcription

The WaveSurfer ASR plugin uses the open-source American English pronunciation dictionary compiled at Carnegie Mellon University [7]. This dictionary uses Arpabet [6] for phonetic transcription. We extracted from the dictionary file `plugins/asr/English/full.dic` the list of phoneme codes actually used. There are 39 of them, as well as two codes for silence `SIL` and short pause `SP`. The list is presented in table 1, where most example words are taken from [6, 7], and International Phonetic Alphabet symbols are typeset using TIPA [2].

In the transcriptions provided with the TIMIT corpus [1], the codes for silences are different, see table 2, and additional phonemes are used, see table 3. In particular, occlusives (B, D, G, P, T, K) are separated into two parts, the closure (BCL, DCL, GCL, PCL, TCL, KCL) and the release (B, D, G, P, T, K).

Some of these additional phonemes are of particular interest to us, as they are allophones, the use of which is optional (it may be dependent on the speaker, the speaking rate, the phonemic context, etc.). They are not used in the dictionaries (including the dictionary provided with TIMIT), but they can be produced by phonological rules.

3.2 Phonological rules

We found a list of some phonological rules used in American English on the web site of the Florida Linguistic Association [9], and other rules in [4]. In both cases, the rules are given using the International Phonetic Alphabet, so we need to convert them into extended Arpabet.

Another problem is that most of the rules require information that is not available in the dictionary of the WaveSurfer ASR plugin: whether a vowel is stress or unstress, syllabic boundaries, etc. We chose to ignore this information, allowing the rule to be applied in more situations than it should be.

Here is a selection of transcribed rules. The syntax is the following:

$$\langle \textit{pattern} \rangle \longrightarrow \langle \textit{replacement} \rangle \; [\; : \langle \textit{left context} \rangle \; \_ \; \langle \textit{right context} \rangle \;]$$

Arpabet code	IPA	example word	transcription
AA	ɑ	father	F AA DH ER
AE	æ	at	AE T
AH	ʌ	hut	HH AH T
AO	ɔ	ought	AO T
AW	aʊ	cow	K AW
AY	aɪ	hide	HH AY D
B	b	be	B IY
CH	tʃ	cheese	CH IY Z
D	d	day	D EY
DH	ð	that	DH AE T
EH	ɛ	red	R EH D
ER	ɝ	hurt	HH ER T
EY	eɪ	ate	EY T
F	f	for	F AO R
G	g	green	G R IY N
HH	h	he	HH IY
IH	ɪ	it	IH T
IY	i	eat	IY T
JH	dʒ	just	JH AH S T
K	k	key	K IY
L	ɫ	late	L EY T
M	m	me	M IY
N	n	no	N OW
NG	ŋ	sing	S IH NG
OW	oʊ	coat	K OW T
OY	ɔɪ	toy	T OY
P	p	pay	P EY
R	r	run	R AH N
S	s	sea	S IY
SH	ʃ	she	SH IY
T	t	tea	T IY
TH	θ	thanks	TH AE NG K S
UH	ʊ	hood	HH UH D
UW	u	two	T UW
V	v	very	V EH R IY
W	w	we	W IY
Y	j	yield	Y IY L D
Z	z	zoo	Z UW
ZH	ʒ	measure	M EH ZH ER

Table 1: Arpabet phonetic transcription

code	name	use
EPI	epenthetic silence	TIMIT
H#	silence	TIMIT
PAU	short pause	TIMIT
SIL	silence	ASR plugin
SP	short pause	ASR plugin

Table 2: Silence codes

code	IPA	name	example word	transcription
AX	ə	schwa	away	AX W EY
AX-H		devoiced schwa	suspect	S AX-H S P EH K T
AXR	ɚ	R-colored schwa	coward	K AW AXR D
BCL	b̚	B closure		
DCL	d̚	D closure		
DX	ɾ	alveolar flap	muddy	M AH DX IY
EL	l̩	syllabic L	bottle	B AO DX EL
EM	m̩	syllabic M	rhythm	R IH DH EM
EN	n̩	syllabic N	button	B AH T EN
ENG	ŋ̩	syllabic NG	Washington	W AO SH ENG T EN
GCL	g̚	G closure		
HV	ɦ	voiced H	ahead	AX HV EH D
IX	ɨ	weak I	using	Y UW Z IX NG
KCL	k̚	K closure		
NX	ɾ̃	nasal flap	winner	W IH NX AXR
PCL	p̚	P closure		
Q	ʔ	glottal stop	uh-oh	Q AH Q OW
TCL	t̚	T closure		
UX	ʉ	fronted U		

Table 3: Arpabet extensions

where the context part starting from the colon is optional. the identifiers ⟨*pattern*⟩, ⟨*replacement*⟩, ⟨*left context*⟩, and ⟨*right context*⟩ represent sequences of phonemes (possibly empty or reduced to a single phoneme), or (except for ⟨*replacement*⟩) several such sequences separated by |, meaning that one of the sequences should be present. Also, in the following list ⟨*vowel*⟩ stands for AA|AE|AH|AO|AW|AX|AXR|AY|EH|ER|EY|IH|IX|IY|OW|OY|UH|UW|UX.

- low/mid vowel reduction
  AA|AE|AH|AO|EH|ER|EY|OW|UH ⟶ AX
- high vowel reduction
  IH|IY|UW ⟶ IX
- R vowel reduction
  ER ⟶ AXR
- syllabic L reduction
  AX L|IX L ⟶ EL
- syllabic M reduction
  AX M|IX M ⟶ EM
- syllabic N reduction
  AX N|IX N ⟶ EN
- syllabic R reduction
  AX R|IX R ⟶ AXR
- flapping
  DCL D|TCL T ⟶ DX : ⟨*vowel*⟩ _ AX|AXR|IX
- H voicing
  HH ⟶ HV : ⟨*vowel*⟩ _ ⟨*vowel*⟩
- syllabe-final T
  T ⟶ Q : _ BCL|CH|DCL|DH|GCL|HH|KCL|M|N|PCL|TCL

3.3 Test data

For test data, we use part of the TIMIT corpus [1]. For each utterance, TIMIT provides four files:

- *name*.`wav`, the audio file;
- *name*.`txt`, the text transcription;
- *name*.`wrd`, the aligned words (we will not use this file);
- *name*.`phn`, the aligned phonemes.

The forced alignment option of the WaveSurfer ASR plugin does not accept the *name*`.txt` as input, because it also contains timestamps and ponctuation. So we created new files *name*`.ort` with timestamps and ponctuation removed.

For initial tests, we selected 4 sentences among the TIMIT core test set:

- Sentence 1 (file `test/dr1/felc0/sx396.wav`):
  The fish began to leap frantically on the surface of the small lake.
  Speaker is a 31-year-old woman from New England.

- Sentence 2 (file `test/dr5/mbpm0/sx137.wav`):
  Tradition requires parental approval for under-age marriage.
  Speaker is a 25-year-old man from Southern USA.

- Sentence 3 (file `test/dr4/fjlm0/sx323.wav`):
  The fog prevented them from arriving on time.
  Speaker is a 23-year-old woman from South Midlands.

- Sentence 4 (file `test/dr7/mgrt0/sx10.wav`):
  Are your grades higher or lower than Nancy's?
  Speaker is a 29-year-old man from Western USA.

4 Experiments

4.1 Software installation and test

We first installed WaveSurfer and the WaveSurfer ASR plugin, with the intent to test it without any phonological rules before proceeding with the implementation of the rules. This apparently simple task took us much more time than expected, until we discovered that, contrarily to what is said in its installation instructions, the plugin cannot run if its Swedish language pack is not installed, even if Swedish is not used.

With both the English and Swedish language packs installed, the plugin runs. Later, we found out that it is possible to modify the plugin so that it uses the English language pack by default, and then the Swedish pack is no longer needed (but the English pack becomes compulsory even if only Swedish is used).

As a first test, we ran the ASR plugin on our four test sentences, first in standard recognition mode, then in forced alignment. We observed that standard recognition performs poorly, with all four sentences resulting in a meaningless sequence of words (see figures 2 to 5).

4.2 When to apply the rules?

We had to face an important choice: at which point of the plugin should we insert our own code to apply phonological rules? In order to decide this, we had to analyse how forced alignment is prepared in the plugin.

The plugin communicates with Julius through command-line arguments and several temporary files, named `/tmp/asrtmp.$$.wav` (the sound data), `/tmp/asrtmp.$$.lst` (which contains only the name of the sound data file), `/tmp/asrtmp.$$.dfa` (an automaton that gives the sequence of words in the sentence) and `/tmp/asrtmp.$$.dict` (a subset of the dictionary restricted to words used in the sentence). Here `$$` stands for the process number, so that different instances of WaveSurfer do not write in the same files.

Ideally, one wants to apply phonological rules on a sequence of phonemes corresponding to the entire sentence, as rules may occur accross word boundaries. But this sequence is constructed by Julius itself, and a quick look into the source code of Julius convinced us that understanding enough of this code to modify it was not doable in a reasonable time.

The second option is to apply the phonological rules to each word independently from the others. This could be done inside the plugin, in the function `TranscriptionToJuliusGrammar` where `/tmp/asrtmp.$$.dict` is constructed. However, since this is our first experience with TCL programming, we prefered to invoke an external program right after the call to this function.

The external program (a Perl script named `pr`, stored as the Julius binary in `plugins/asr/bin/`), takes as arguments the name of the temporary dictionary and of the file containing the rules themselves, renames the dictionary, and creates a new dictionary that contains all the pronunciation variants obtained by applying recursively (and optionally) the rules until no new variant is found, as is done in [4][1].

The rules themselves are stored in a separate file, added in the English language pack: `plugins/asr/English/rules`. This file has a simple text format that makes it easily customizable.

4.3 User interface

Some TCL code was added in the plugin, mimicking existing code, to add a checkbox in the Properties dialog of the ASR pane (see figure 1), so that it is easy to perform alignment with and without phonological rules.

[1]The program does not attempt to check that this algorithm terminates. It is easy to design ad hoc rules that will cause it to go into an infinite loop. This does not happen with real phonological rules, which usually do not increase the length of the phoneme sequence.

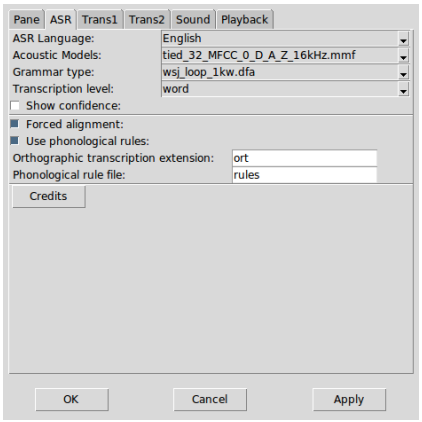

Figure 1: Properties dialog of the modified ASR plugin

5 Results

5.1 Using real rules

We first tried the first rule: vowels AA, AE, AH, etc. may be reduced to a schwa AX. The dictionary was correctly augmented with variants, but recognition then failed with a lengthy error message from Julius.

The reason for this was easily found: the language model used by the WaveSurfer ASR plugin knows only the 39 phonemes of table 1, plus SIL and SP. To use our full set of rules, we would need to train a completely new language model.

5.2 Using modified rules

Since we did not have time for building a new language model, we resorted to replace the rules with modified ones that use only those 39 phonemes. For some of the rules such as flapping, this does not make sense, so we kept only two kinds of rules:

- vowel reduction, using AH and IH instead of AX and IX;
- syllabic L, M, N, R reduction, using AH L, AH M, AH N instead of EL, EM, EN, and ER instead of AXR.

This results in incorrect pronunciations, but that may be in certain cases closer from the real pronunciation than the standard one given by the dictionary.

We ran our modified plugin on our four sentences. The results are shown in figures 2 to 5.

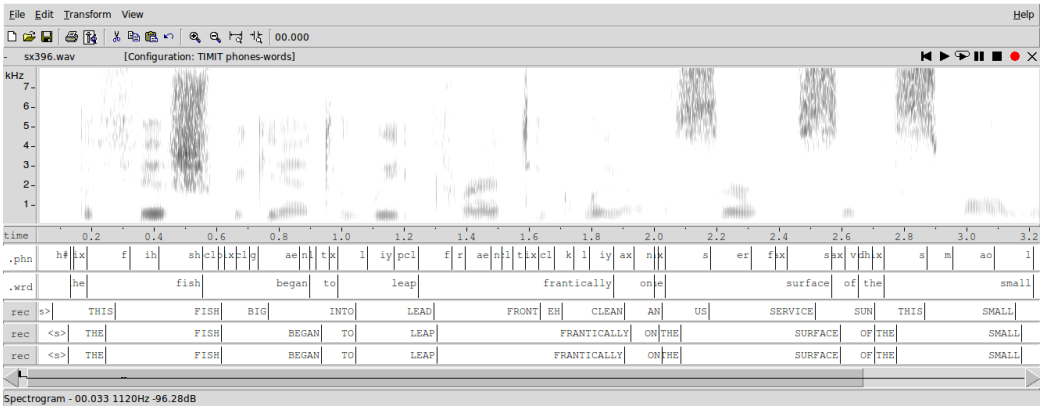

Figure 2: Forced alignment for sentence 1

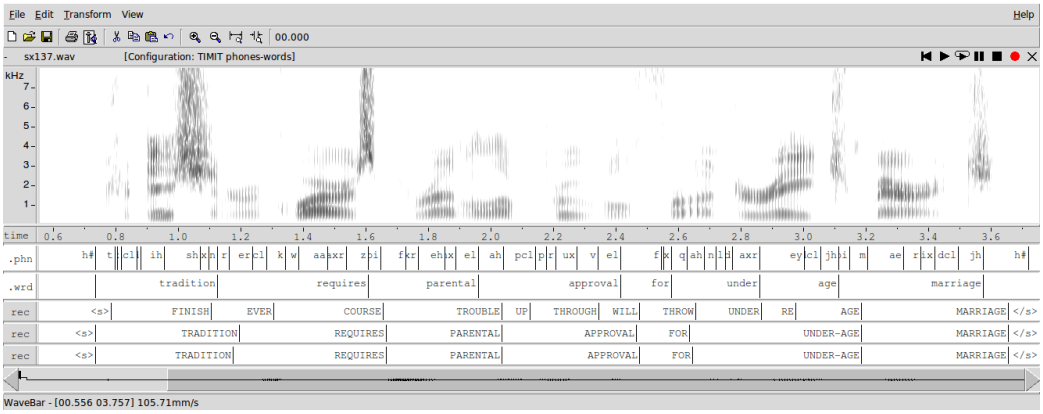

Figure 3: Forced alignment for sentence 2

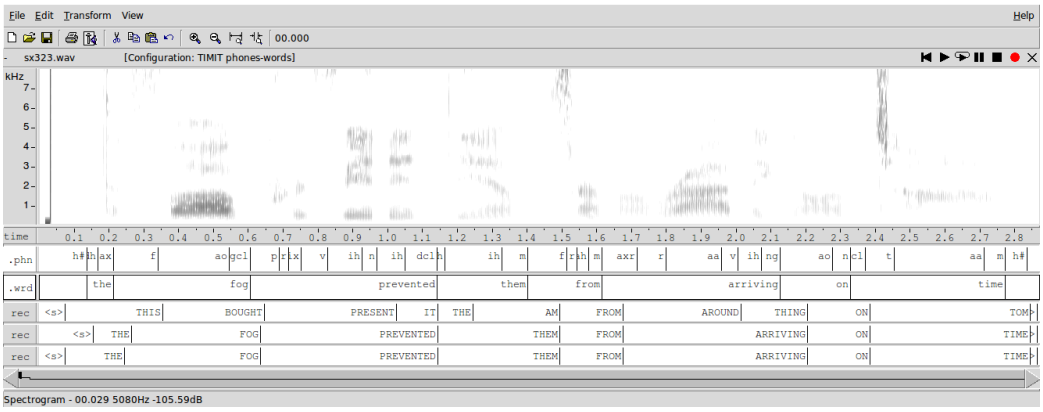

Figure 4: Forced alignment for sentence 3

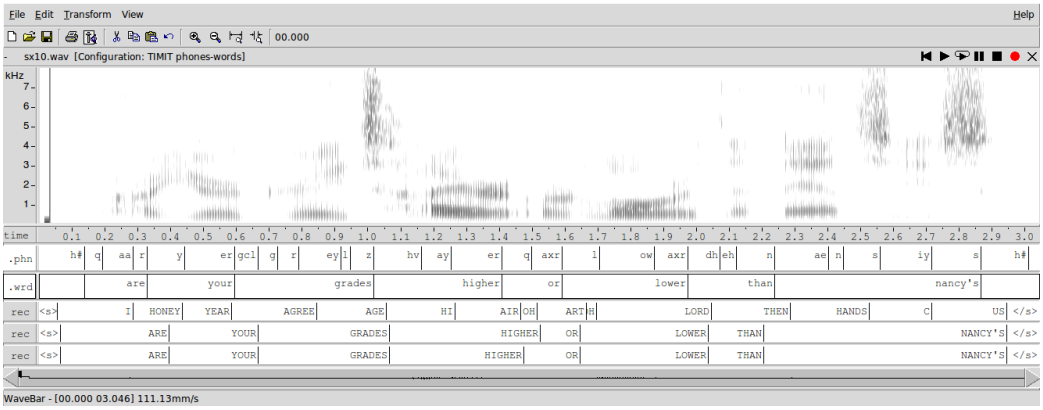

Figure 5: Forced alignment for sentence 4

The figures show the WaveSurfer window with several panes:

- a spectrogram,
- a time axis,
- the aligned phonemes provided by TIMIT,
- the aligned word provided by TIMIT,
- the output of the ASR plugin in standard recognition mode,
- the output of the ASR plugin in forced alignment mode,
- the output of the ASR plugin in forced alignment mode with phonological rules.

As already noted, the output of the plugin in standard recognition mode is often very far from the real sequence, and does not seem usable. The last two panes are mostly identical. The alignment is different from the one proposed by TIMIT, and very often the end of the phoneme is too late by up to 0.05s. This is clear for instance on the word *fish* in sentence 1, where the end of SH is clear in the spectrogram.

However they are not completely identical, and the influence of the rules can be seen on a few words. In sentence 1, the words *on the* seem better aligned. In sentence 2, the syllabic N and L in the words *tradition* and *approval* are aligned differently, but it is not clearly better. In sentence 3, the initial *the* starts much earlier, which is quite surprising as no rule affects the first phoneme DH. In sentence 4, the ER in *higher* is better aligned, but still too late, probably because the following glottal stop Q is not correctly modeled.

6 Discussion and Conclusions

We implemented the use of phonological rules for forced alignment in the WaveSurfer ASR plugin. The results are currently not very convincing: alignment is slightly modified, sometimes but not always improved, but still quite different from the reference alignment in TIMIT.

The key to improve it does not seem to be in the phonological rules themselves, but in the language model, which should use a larger set of phonemes, with finer distinctions. This would in turn allow to implement a larger set of phonological rules. Also, stress should be taken into account.

Further improvements could probably be achieved by re-implementing phonological rules inside Julius, in order to make them also operate across adjacent words. The user interface could be kept, with the call of an external program replaced with an additional argument passed to Julius. At that stage, it would probably be interesting to follow a different approach: instead of recursively applying the rules to generate a potentially long list of variants, one could use a finite automaton to encode all variants simultaneously. Of course, according to theory arbitrary context-sensitive rules cannot be expressed by a finite automaton, but the particular rules that are used here create only a finite list of variants so the automation always exists.

References

[1] J. Garofolo et al., *TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1*. Web Download. Philadelphia: Linguistic Data Consortium, 1993. [`https://catalog.ldc.upenn.edu/LDC93S1`]

[2] F. Rei, TIPA: A System for Processing Phonetic Symbols in LaTeX, *TUGboat* 17, 1996.

[3] G. Salvi and N. Vanhainen, The WaveSurfer Automatic Speech Recognition Plugin, *Proceedings of LREC*, Reykjavik, Iceland, 2014. [`http://speech.kth.se/asr/`]

[4] G. Tajchman, D. Jurafsky, and E. Fosler, *Learning Phonological Rule Probabilities from Speech Corpora with Exploratory Computational Phonology*, ACL '95, 1995.

[5] V. Zue, S. Seneff, and J. R. Glass, Speech database development at MIT: Timit and beyond, *Speech Communication* 9 (1990), 351–356.

[6] *Arpabet*, Wikipedia. [`http://en.wikipedia.org/wiki/Arpabet`]

[7] *The CMU Pronouncing Dictionary*, Carnegie Mellon University. [http://www.speech.cs.cmu.edu/cgi-bin/cmudict]

[8] *Open-Source Large Vocabulary CSR Engine Julius*, Nagoya Institute of Technology. [http://julius.osdn.jp/en_index.php]

[9] *Phonological Rule Inventory*, Florida Linguistic Association. [http://floridalinguistics.com/?page_id=587]