Manual: The Gradient Lexicon and Phonology Learner

Claire Moore-Cantwell

November 2, 2022

1 Quick Start

2 Details of Theories Implemented

2.1 Preliminaries

2.1.1 Theories of Eval

2.1.2 Perceptron Learning

2.2 UseListed

UseListed is a theory that many researchers implicitly or explicitly use as a default approach to exceptionality in phonology, but as far as I know no learning model has been developed. The theory can easily be summarized as "We memorize exceptions."

This theory assumes two things:

1. We memorize morphologically complex words, at least sometimes

2. We can therefore produce morphologically complex forms in two ways:

- **Composed** forms are created by accessing multiple morphemes and realizing them together according to the morphological and phonological grammar
- **Listed** forms are accessed whole, and realized according to their lexical entry and the phonological grammar

A couple of examples:

In Tagalog, morphemes often undergo 'nasal substitution' in which the final nasal of a prefix coalesces with the initial consonant of the root, forming a single sound.

(1)	a.	**d**iníg *audible*	/pa**ŋ**+**d**iníg/	→	pa**n-d**iníg *sense of hearing*	ASSIMILATION
	b.	**d**aláŋin *prayer*	/i+pa**ŋ**+**d**aláŋin+in/	→	ʔi-pa-**n**aláŋin-in *to pray*	SUBSTITUTION

These examples come from **?**. In (1a), the underlying velar nasal assimilates in place to the following stop, but in (1b), the two sounds have completely merged, leaving an [n] with the nasality of the [ŋ] and the place of the [d].

As the example suggests, the phonological shape of the individual words cannot completely predict whether the nasal will substitute or just assimilate. There are lexical trends, but no categorical rules. Importantly, individual words do not vary, the variation is entirely from word to word.

A UseListed approach to this pattern would say that both pa**n-d**iníg and ʔi-pa-**n**aláŋin-in are memorized as their own lexical entries. When speakers want to say the meaning *'sense of hearing'*, or *'to pray'*, they access those meanings directly in their lexicon, and produce them according to the idiosyncratic phonological form that is listed - one with substitution, and one without.

A second example:

English comparatives come in two forms, the 'periphrastic', using *more*, and the 'morphological', using *-er*. Both are available for most adjectives: *fouler* and *more foul* are about equally attested in a corpus, for example. However, higher-frequency adjectives exhibit idiosyncratic preferences **?**.

(2) a. simpler (96%) $\gg$ more simple (4%)

b. more stable (98%) $\gg$ stabler (2%)

While a variety of phonological factors condition the choice between these two versions of the comparative, *simple* and *stable* are similar on all relevant dimensions[1]. English speakers therefore must memorize that the meaning *simple* + COMPARATIVE is pronounced with *-er*, while *stable* + COMPARATIVE is pronounced with *more*.

Unlike in the case of Tagalog nasal substitution, many individual words do vary, and low-frequency words seem to follow the predictions of a probabilistic grammar. Even extreme cases like those in (2) still exhibit the minority form occasionally. One was to imagine what is happening here is that speakers have memorized the majority form for both words, but every so often they fail to use that memorized form and compose the comparative on the fly instead.

If we have both a **Composed** form and a **Listed** form available, there are many different ways we could decide between them. GLaPL can learn using any of these.

Option 1: Always use the listed form, if available. OR use the listed form with some static probability

Option 2: Do whatever is easier. If it is easy to find the listed form, use it, but if it is easier to compose, use the composed form.

[1] They are 2-syllable words with initial stress, ending in [l], and are about the same lexical frequency.

Option 3: Directly compare the Composed and Listed derivations in the same tableau.

To do **Option 1**, set `p_useListed` to 1 or less. That will be the probability that a PREDICT step will use the listed form, if available.

To do **Option 2**, set `p_useListed` to 2. There are many possible ways to implement 'easier' here, but for now the implementation is based on lexical frequency. The probability that the listed form will be chosen is related to both the frequency of the listed form itself and the frequency of the lowest-frequency morpheme in the composed form.

$$P(listed) = \frac{freq_{listed}}{freq_{listed} + min(\{freq_{composed1}, freq_{composed2} \ldots freq_{composedx}\})}$$

Hay 2003, Forster and Chambers 1973, Whaley 1987

2.3 Lexically Indexed Constraints

2.4 UR-constraints

2.5 Representational Strength Theory

2.6 Gradient Symbolic Representations

3 Supplementary details

3.1 How Tableaux are constructed

Tableaux are constructed in real time during learning. At each learning iteration, a tableau is constructed for that input-output pair. Here is a breakdown of how the `Grammar.makeTableau()` function operates.

`Grammar.makeTableau()`'s first argument is a **datum**, which is an entry in `Grammar.trainingData.learnData`, a list.

(3) **datum:** [[$\texttt{lexeme}_1$, $\texttt{lexeme}_2$, $\ldots\texttt{lexeme}_\texttt{n}$], surface string, input]

The surface string and input (also a string) both come directly from the input file. The surface string comes either from your 'candidate' column, or from your 'surface' column, if you have one. The input string comes from your 'input' column.

datum's first entry is a list of `lexeme` objects involved in this learning datum. Note that if you are using UseListed, these won't always be the actual lexemes used to generate the tableau - the function may try to use the listed form instead.

The first step in creating a tableau is to generate faithful candidates from the input lexemes. In most cases, this will be a single faithful candidate, but not always. The inner function `lexemesToFaithCands()` accomplishes this.

In order to directly test `lexemesToFaithCands()` behaviour on a list of lexemes, run `makeTableau()` with the option `textFcs=True`.

```
import learner as l
g = l.Grammar("lexemes_sampler.txt", l.Features("features.txt"))
g.makeTableau([lexemes],testFcs=True)
g.makeTableau([l.exlex_petit(),l.exlex_ami()],rich=True,testFcs=True)
```

You can find "lexemes_sampler.txt" in the manual_examples folder

3.2 Dealing with hidden Structure

Hidden structure problems can arise in any theory you are using, but UR-constraint models, UseListed with the UseListed constraint, and GSR models with segment blending all require hidden structure to operate. GLaPL follows the principles of Jarosz (2013)'s Expectation-Maximization algorithm ot learn hidden structure.

Let's look at an example using UseListed. UR-constraints would function similarly.

(4) Tableau for made-up multi-morphemic word, based on Pater et al, 2012.

			*VTV	Ident-Voice	UseListed			
		weights:	1	2	1	H	p	
Composed	a.	/mot/+/a/	→ mota	1	0	1	-2	0.11
	b.	/mot/+/a/	→ moda	0	1	1	-3	0.04
Listed	c.	/moda/	→ mota	1	1	0	-3	0.04
	d.	/moda/	→ moda	0	0	0	0	0.81

3.3 Simulating Gen

3.4 Applying constraints

4 Input file details

5 Classes and Methods