Michal Shuvi 313450363

Learning rule-based phonology from voice sound signals

Abstract

As part of language acquisition, the child receives a lot of sound signals, he interprets those speech sounds and associates them to sounds he has already heard. After creating some cognitive representations to these speech sounds, he starts to form cognitive system which describes the way sounds function within the heard language (i.e. phonology of the language) to encode meaning in the future. In this paper we try to imitate this learning procedure artificially, using a computer code. This learning process works as follows. It starts with a recorded speech sounds, which it classifies using MDL classifier into main groups that represent language components. Then it uses MDL-based algorithm to learn some simple phonological rules, like vowel harmony, from the processed input.

1. Introduction

This paper offers a learning model which learns rule-based phonology from speech sound signals. This work begins with a pure wave speech sound of vowels only, for simplicity, and ends with a modest phonology system. This paper crosses two main linguistics study fields, phonetics and phonology.

Learning and understanding natural language starts with speech sound signals, infants and adults receive these signals and analyze their physics-related components into some cognitive representations, the branch of linguistics which studies the sounds of human speech is *phonetics*. Speech sound signal consists of distinctive frequency components which are called **formants**, the two formants with the lowest frequency are called F1 and F2, these two are enough to disambiguate a vowel, we will discuss this later in the paper.

The next step of acquiring a natural language is learning its *phonology*, a cognitive system composed of representations, rules, and principles responsible for the organization of speech sounds in the language. We use phonology features to represent vowels and consonants and to create phonological rules. In this work we will focus mainly on the representation of the language elements (i.e. vowels) and learning some very simple phonological rules according to the input voice signals.

Main principle in this paper is Minimal Description Length (MDL) which is used a couple of times throughout the paper and will be detailed later.

2. Learner Pipeline

In this section we will discuss the main pipeline which the work is based on, the phonology learner from pure signals, we will examine its parts and describe the whole process.

The input of the learner is a speech voice signal composed of vowels only, i.e. recorded "Mars language" sentences. In this work we assume that the signal is pre-marked with segment (in this case, ש vowel) boundaries and word boundaries.

The output, our main goal, includes the UR representation of the language grammar:

a. The learned phonology rules, e.g $[+back] \rightarrow -back]/[+low]$ __.
b. The UR representation of the lexicon.

As represented in figure 1 the work contains four main phases:

1. Phonetic Parser - its purpose is to extract the two lowest frequency formants: f1 and f2 for each vowel.
2. Phonetic Cluster - based on the vowel (f1, f2) map, the cluster should divide the f1-f2 space into centroids, each centroid represents vowel in the language, the prototype of the vowel should be the center of the centroid.
3. Features Tagger - this step is the point in which our phonetic knowledge turns into phonology knowledge. The tagger gives each vowel prototype phonology features and their suitable coefficients based on f1, f2 frequencies and relation between prototypes.
4. Phonology Learner - its goal is to learn phonological rules using MDL, Minimal Description Length, which will be explained in detail.

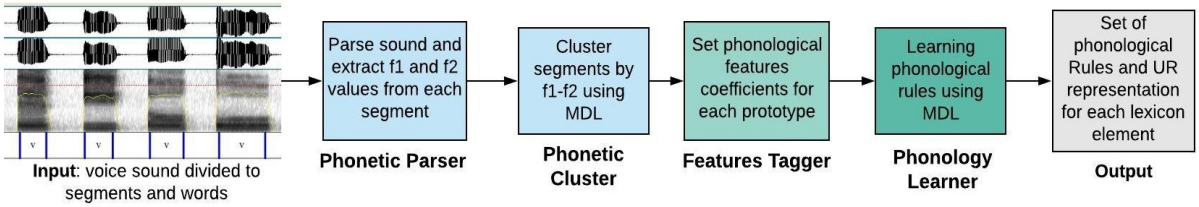

Figure 1: the learner pipeline consists of four processes. the input is recorded sentences and Praat textGrid, the output is UR representation of grammar - lexicon and rules.

2.1 Phonetic Parser

In physics, sound is vibration that typically propagates as an audible wave of pressure, through a transmission medium such as a gas, liquid or solid. Humans receive those voice waves and analyze them. Our learner's input is a speech voice signal and the very first task is to interpret those waves and frequencies into units, these units represent components of some natural language, which could be consonants or vowels. In this paper we will focus on vowels.

In order to distinguish between vowels, Daniel Jones 1956 set a group of eight cardinal vowels, such that every vowel can be represented according to them. he also gave articulatory criteria of the cardinal vowels, we will focus on two of them, tongue height (the vowel height) and tongue back (the vowel back). e.g. the vowel [i] like in the English word "seat" is the highest and the most front of all vowels, and on the other edge of the Scala there is the vowel [□] like in the English word "father" which is the lowest and the most back of all vowels.

As discussed earlier, vowels can be represented using feature vector which contains the two low frequency formants, f1 and f2. F1 is used to calculate the vowel height, the lower f1 is the higher the vowel is. F2 is used to calculate the vowel back, the lowest f2-f1 (difference) is the back vowel is.

One platform for manually analyzing speech sound signal is the computer program *Praat* by Boersma, 2001. Jadoul, Thompson and de Boer 2018 present *Parselmouth* an open-source python interface to Praat. In this work parselmouth was used to extract the first two formants from each speech element that was pre-marked using Praat TextGrid. Each vowel signal is sampled a few times during its time window, the relevant formant is calculated for each time stamp, and the final formant value is calculated as the mean of all the vowel previous results.

The output of this phase is a list of speech segments and their f1, f2 values in Hz. The next step is to find the segments that are most likely to be of the same vowel, the clustering phase.

2.2 Phonetic features cluster

Phonetic features' frequencies of the same vowel may differ due to differences between speakers' physiological structure of the sound production track, acoustic disturbances or even intonation of the speaker. But as listeners we still know to which category the vowel belongs to. Khul 1991 introduces the notion of *a Perceptual Magnet*, the prototype of a phonetic category functions like a perceptual magnet to other category members. It means that listeners recognize the suitable category of a vowel by calculating the distance from it to every (vowel) prototype of the language, after that they choose the closest prototype, and by this, the prototype category.

This phase input is a sequence of vowels with their (f1, f2) features, its goal is to find the phonetic categories of the data (i.e. the language) and their prototypes. We can achieve this by dividing the input data, (f1, f2) points, into centroids, each centroid represents category and should contain all of its members, the category prototype will be the center of the centroid.

2.2.1 Minimal Description Length for categories learner

For the task of learning phonetic categories, we use Minimal Description Length (MDL; Rissanen 1978) evaluation metric, which incorporates both the idea of grammar simplicity (or economy) and that of restrictiveness (or how easy it is for the grammar to capture the data). For our goals, the grammar, notated by G, would be the set of phonetic prototypes in the language. For grammar economy we will consider G as the number of elements in it, how many prototypes the data has, which will be notated as |G|. Restrictiveness will be thought of in terms of how simple it is to describe the data, D, given the grammar, G, a description that we will notate as D:G. In our case, D is the given (f1, f2) points that represent the input vowels, D:G is a set of Euclid distances between each point to the closest prototype. |D:G| would be the sum of those distances[1].

$$MDL\ metric: If\ G\ and\ G'\ can\ both\ generate\ the\ data\ D, and\ if\ |G| + |D:G| < |G'| + |D:G'|, prefer\ G\ to\ G'.$$

Since the more prototypes we have in our inventory (bigger |G|), the smaller the average distance from each point to its nearest prototypes (smaller |D:G|), And vice versa, The optimal solution will come from the balanced kind of learning in MDL.

Let us examine the MDL intuition for this case by an example. let's assume that there are some points in a 2D space like in figure 2, we can clearly see that there are two main point groups, we will notate the left one by A and the right one by B. If G is the empty set, then the distances in D:G are not defined so $|D:G| = \infty$. If G' contains one point, μ, supposed that it is from A, then μ is closer to A points than to B points. We can create additional grammar, G'' by adding another segment to G' from group B, ϕ.
Now $|G''| > |G'|$ but $|D:G''| < |D:G'|$, because ϕ shortened the distances to B points.

[1] In the algorithm we multiply |D:G| by some constant, , because |G| has different units than |D:G|. |G| is a natural number which counts the number of prototypes, and |D:G| is a sum of Euclid distances in Hz scale, so , should normalize those two.

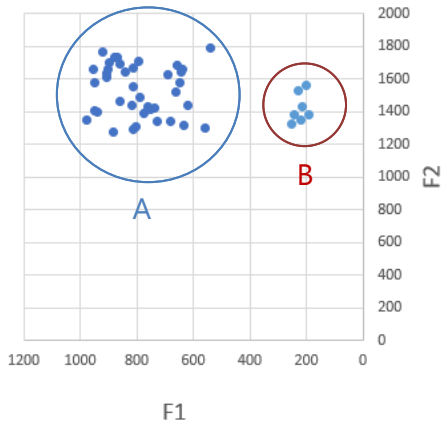

Figure 2: Two main groups of sound element, A and B.

2.2.2 Simulated Annealing

How can we find the optimal grammar that would minimize $|G| + |D:G|$?
For this task we will use a probabilistic technique, **Simulated Annealing**. It is a metaheuristic to approximate global optimization in a large search space for an optimization problem. The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.

The simulated annealing algorithm in our case works as follows. At each time step, the algorithm randomly selects a neighbor G' to the current G, it can add a prototype to G or to remove one. After that the algorithm measures the neighbor G' quality, $|G'| + |D:G'|$ as described above, and then decides to move to it or to stay with the current G based on either one of two probabilities between which it chooses on the basis of the fact that the neighbor G' is better or worse than the current G. During the search, the temperature is progressively decreased from an initial positive value to some predefined threshold and affects the two probabilities: at each step, the probability of moving to a better new solution is either kept to 1 or is changed towards a positive value; on the other hand, the probability of moving to a worse neighbor grammar is progressively changed towards zero, which means that at the beginning the algorithm will take more risks and move to worse G' and with time will take less and less risks. In addition, the algorithm has a cooling rate, a number between 0 and 1 that sets the rate of the algorithm, how slow it cools down and reaches the threshold.

In this work code we started with an empty set as grammar, G, 1000 as initial temperature, 0.9995 cooling rate and 10^{-6} as threshold, and the data was all the recorded vowels with their (f1, f2) frequencies.

2.2.3 Input and Results

The input data for this stage of the code were five Hebrew vowels [a], [e], [i], [o] and [u]. Each vowel was recorded four different times by the same speaker and went through the first phase of the pipeline. The code results are shown in figure 3 as a scatter chart, the vowel notations of "a", "e"… are not part of the MDL cluster real results but written for readability.

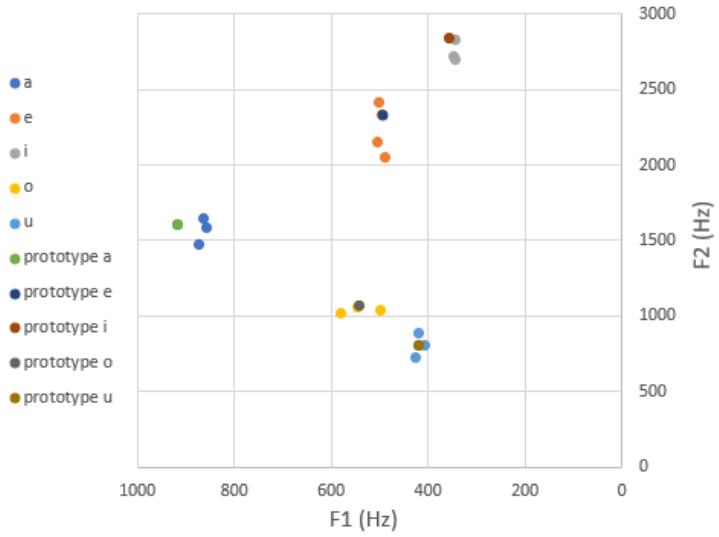

Figure 3

As we can see, the MDL cluster returns five categories, each one represents a real phonetic category in Hebrew. We were able to divide the f1 f2 space into the language's estimated categories. In the next phase we will try to set some phonological features for every category so we could better understand the mental representation of each category and to get closer to develop some small phonological system.

2.3 Phonology Features Tagger

This is the point where we leave phonetics and passing to phonology. In phonology, segments are not indivisible units, but are composed of features. A feature system must be able to describe classes of sounds, specify phonetic detail and distinguish phonemes.

Because our inputs are vowels only, we will focus on vowel features, and specifically on three of them: high, low and back. Each feature is preceded by a coefficient: + or -. We will assume that those features exist and known in the language and every vowel must have some value to every feature.

To give the right coefficient to each phonetic category we will use means and relations between the phonetic category prototypes. As described above, formant f1 is responsible to determine the height of the vowel, so it influences low and high features coefficients. The finding mean process will work as follows. We will find the prototype with highest f1 value, σ, and the one with the lowest f1 value, β. After that we will choose the vowel with lowest frequency of f1 from σ category[2] σ_{min}, and the highest one from β category, β_{max}.

$$mean_{low\_feature} = \frac{\sigma_{min} + \beta_{max}}{2}$$

if a prototype is higher than $mean_{low\_feature}$, then it is [+low], [-low] otherwise. After checking this condition for every prototype, we will tag each category member with the right coefficient to the low feature. Because [+low] and [+high] conflict, we will set their high feature into (-) and omit them

[2] the main purpose was to find the lowest bound of the category so in more complex situations we need to make sure the point is not exceptional

for the next step, setting the high feature values. We will find the mean frequency for the high feature the same way we did with low feature. Prototypes that are lower than $mean_{high\_feature}$ are [+high], [-high] otherwise. Vowels between $mean_{low\_feature}$ and $mean_{high\_feature}$ are [-high, -low], i.e. they are in the middle of the height scala.

As mentioned above, (f2-f1) sets the vowel back, so we will repeat the mean finding process on f2-f1 values. Prototypes that are lower than $mean_{back\_feature}$ are [+back], [-back] otherwise.

To sum up this phase, we started with formants frequencies of phonetic categories as inputs, and ended up with phonological description for each such category, i.e. category and its phonological features values. Results shown in figure 4, prototypes.

prototypes features	**1** **(917, 1607)** **(a)**	**2** **(494, 2338)** **(e)**	**3** **(356, 2845)** **(i)**	**4** **(543, 1073)** **(o)**	**5** **(917, 1067)** **(u)**
High	-	-	+	-	+
Low	+	-	-	-	-
Back	+	-	-	+	+

***Figure 4**: Each prototype and its f1, f2 values, in addition the orthography of the vowels is the same as in figure 3. phonological feature coefficient is the code output.*

2.4 Rule-based Phonology Learner

In the last phase we will try to learn phonological rules of vowels in the "Mars" language we have created. This phase input is a sequence of words that consist of vowels, each vowel represented by [high, low, back] coefficients, these vowels are PRs. the desirable output will be a set of phonological rules and for each vowel in a word - which phonological features are used for storage of its UR.

The main idea is that by learning a phonological rules we economize in features that are needed for storage of URs. For example, if a language has a rule in which a word must end with an [+high] vowel. We have to store the rule but we don't need to store high feature coefficient for every last vowel in every word in the language, as we can see it is very efficient. On the other hand, we need to be careful of under-generalization which leads to high number rules and close to zero used features for URs storage. That sense takes us to MDL criteria as described in phonetic feature cluster section. In this phase the grammar consists of rules and lexicon, lexicon is all different words of the input words, or in other words, set of the input words, the Data is the input of this phase, i.e. words as represented in the PR. For grammar economy we will consider G like it is sitting in a computer memory in terms of how many bits takes after a given encoding, the number of bits will be noted as $|G|$. We shall describe the phonological representation and the encoding method of grammar in order to explain clearly the MDL score, firstly for the lexicon and phonological rules afterwards.

2.4.1 Vowel representation

A vowel is represented as a set of features and their coefficients, like in figure 4 above, in other words as feature bundle. there are three features and two optional coefficients, so five elements in total. To get the number of bits needed for representing each element we log_2 the number of elements (i.e. five), the result is three bits. Each element is mapped to a code in bits, an example for

a possible conversion table appears in figure 5.

Symbol	Code
high	000
low	001
back	010
+	011
-	100

***Figure 5**: Conversion table for vowels*

Lexicon is encoded as a sequence of vowels separated into words, each vowel code is feature coefficient pairs, each one encoded as appeared in figure 4.

2.4.2 Rule representation

Phonological rules represented as feature bundles as well. The general form of rules is as follows, A, B are feature bundles, in this paper, they cannot be the empty set. X and Y cannot be a word edge and at most one of them can be the empty set, figure 6.

$$\underbrace{A}_{focus} \rightarrow \underbrace{B}_{change} \; / \; \underbrace{X}_{\substack{left \\ context}} \_\_ \; \underbrace{Y}_{\substack{right \\ context}}$$

***Figure 6**: Rule format*

As presented in *Rasin, Berger, Lan and Katzir (2018)*, phonological rule can be written in string notation with delimiters marked with # with subscripts, and can be encoded with fixed and equal length codes. Determining the length in bits of a single phonological rule for the purpose of MDL is done by using a conversion table that contains the codes for the possible elements within a rule. It is very similar to the table in Figure 5, except for the added delimiters.

Symbol	Code
high	000
low	001
back	010
+	011
-	100
$\#_{rc}\ (rule\ component)$	101
$\#_{f}\ (feature)$	110

***Figure 7**: Conversion table for rules*

Using the conversion table in figure 7 we can encode a phonological rule by converting each element in the string representation into bit and concatenating these codes. The delimiters are used to ensure unique readability, a feature in a bundle ends with $\#_f$ and a rule component ends with $\#_{rc}$ (in terms of notations in figure 6, after each one of A, B, X and Y).

Let us examine an example of a vowel harmony rule.

a. Textbook notation

$$[-high] \rightarrow [+high] \; / \; [+high, -back] \_\_$$

b. String notation

$$-high\#_{rc} + high\#_{rc} + high\#_{f} - back\#_{rc}\#_{rc}$$

c. bit representation

$$\underbrace{100}_{-}\ \underbrace{000}_{high}\ \underbrace{101}_{\#_{rc}}\ \underbrace{011}_{+}\ \underbrace{000}_{high}\ \underbrace{101}_{\#_{rc}}\ \underbrace{011}_{+}\ \underbrace{000}_{high}\ \underbrace{110}_{\#_{f}}\ \underbrace{100}_{-}\ \underbrace{010}_{back}\ \underbrace{101}_{\#_{rc}}\ \underbrace{101}_{\#_{rc}}$$

After understanding the grammar encoding, we know how to calculate |G|, it would be |lexicon| + |rules|, each one is the number of bits needed for representation as described above. We should also discuss D:G, which in this case is a sequence of the lexicon's words. Calculating |D:G| is done by finding the right word in the lexicon and using its representation and encoding.

2.4.1 Phonology Learner Algorithm Flow

Now we shall discuss the algorithm flow, which its main flow is quite similar to the one described in section 2, again we will use Simulated Annealing for finding the global minimum of |G|+|D:G|.
We start with an empty set of rules, and our lexicon set contains all of the different input words, each vowel is fully represented, which means that all the features are needed for the representation. In every timestamp we will find a neighbor grammar, the neighbor will differ in the rules set because the input isn't changing. A neighbor can have one more rule or delete a rule, it can change a rule of the current grammar as well by adding or deleting a feature from one of the rule components: focus, change, left context or right context.
After finding a neighbor, the algorithm applies each rule on the lexicon as described in the following process. In every word the algorithm looks for the suitable environment according to right and left contexts and makes sure the change component features are included in the vowel between left context and right context, we focus on the change features and not on the focus features because the input words are PRs, it means we do not know what is the vowel UR but we do know its PR, i.e. after a rule is applied, so we try to "reverse" the rule. We can omit the features that included in the rule's change component from this middle vowel and add those in the focus component, after making those changes we have created an estimated UR of the middle vowel and we can restore the full representation using the compatible rule and the "UR". An ultimate grammar would have small amount of features for vowels UR, but will economize in number of rules as well. after applying the rules, the algorithm calculates |G| by creating the representation for every rule and encoding each of them as described above, the sum of all rules bits is |rules|. |lexicon| is the sum of bits of every word as we already mentioned, only after applying the rules we need to recalculate its size due to vowels feature changes, some of them may need less features for representation. After calculating |G| = |rules| + |lexicon|, we shall calculate |D:G| as we already discussed. Finally we sum the results up and get |G|+|D:G|, the MDL score. Using the Simulated Annealing technique we reach the global minimum of the MDL score as we have already seen.

The output of this phase is a grammar which includes the rules set as learned from the input words, and the lexicon after applying the rules, so each vowel in a word has its phonological features that are used for storage of its UR.

2.4.3 Inputs and Results

Let's examine a few examples, we will focus on [a], [e], [i], [o], [u] vowels as in Hebrew, and the words consist of these vowels only.

i. Input: words which consist of "i" vowel only: "ii", "iii"…
Goal: we will try to learn the vowel harmony rule "i", which means that we want to save the full phonological representation of the first vowel of a word, and every other vowel in the word will copy its features values. This learner cannot learn a rule without an environment (i.e. without both right context and left context) or without a focus or and a change, for simplicity and to avoid some edge cases.
Learned Rule: $[+back] \rightarrow [+high, -low, -back]$ / $[+high]$ __
Needed features for UR representation of vowels in word: let's take "ii" for example, the first vowel has all features values, for the second we only need [+back] (because the non-empty focus constraint)

ii. input: "iii" 5 times, "uuu" 4 times, "eii" one time, "ei" one time, "ouu" one time, "ou" 2 times.
Goal: learning a vowel harmony rule $[-high] \rightarrow [+high]/[+high]$ __.
Learned Rule: $[-high] \rightarrow [+high, -low]/[-low]$__
We can see that the rule is more generalized than the goal rule because all input vowels are [-low], it chooses this feature to economize in the needed features for UR representation of all of them.
Needed features for UR representation of vowels in word:
"iii": first "i" - needed features: all, second and third "i"s' needed features: [-back, -high];
"uuu": first "u" - needed features: all, second and third "u"s' needed features: [+back, -high];
"eii" - needed features for "e": all, both "i"s needed features [-back, -high].
It is important to notice that "___" can represent more than one vowel distance.

3. References

Daniel Jones. 1956. *An outline of English phonetics.*

Boersma, P. 2001. *PRAAT, a system for doing phonetics by computer*. Glot International, 5, 341–345

Yannick Jadoul, Bill Thompson, Bart de Boer. 2018. *Introducing Parselmouth: A Python Interface to Praat*.

Kuhl, P. K. (1991). *Human adults and human infants show a "perceptual magnet effect" for the prototypes of speech categories, monkeys do not*. Perception & Psychophysics, 50, 93–107.

Rissanen, Jorma. 1978. *Modeling by shortest data description*. Automatica 14:465– 471.

Rasin, Berger, Lan and Katzir. 2018. *Learning rule-based morpho-phonology.*