Boersma’s Gradual Learning Algorithm

Today’s focus is on GLA as learning algorithm. We’ll discuss stochastic OT as a model of variable grammar later.

(1) Simplifying assumptions
- Child knows input-output mappings (doesn’t have to figure out what inputs are)
- Child knows or has already learned the constraint set
- Procedure exists for calculating winning candidate given input and ranking

(2) Advantages over Constraint Demotion
- Robust to errors, because gradual
- Can handle variation
- Can realistically model learning stages

(3) Background: stochastic constraints
(Boersma 1998,1 Hayes & MacEachern 19982)
For Boersma, the ranking of constraints is neither fixed nor freely variable, but probabilistic.
- Each constraint in an individual’s grammar has a ranking value, given in arbitrary units.
- In each utterance, the speaker generates selection points (“disharmony” for Boersma) for each constraint
- selectionPoint = rankingValue + rankingSpreading * z
- rankingSpreading (“noise” in OTSoft) is some constant, typically 2
- z is a Gaussian random variable with mean 0 and standard deviation 1.

Gaussian random variables are what give us “bell curves” (normal distributions):

Example of selection points for 10 runs of 2-constraint grammar (Boersma p. 331)

<table>
<thead>
<tr>
<th>trial</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>50.5</td>
<td>51.2</td>
<td>50.2</td>
<td>49.1</td>
<td>52.9</td>
<td>52.9</td>
<td>52.7</td>
<td>53.8</td>
<td>55.4</td>
<td>54.3</td>
</tr>
<tr>
<td>(ranking value = 50)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>50.8</td>
<td>48.3</td>
<td>50.7</td>
<td>51.2</td>
<td>48.9</td>
<td>48.8</td>
<td>48.2</td>
<td>50.3</td>
<td>48.1</td>
<td>48.7</td>
</tr>
<tr>
<td>(ranking value = 50)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Constraints are ranked, for the utterance in question, according to their selection points. We can call this the “effective ranking”.

(4) We can think of each constraint as associated with a probability density function centered on the ranking value:

How it works

(5) Start with some default set of rankings values.

Perhaps all ranking values are 100, or perhaps Markedness constraints start at 500, Faithfulness at 100.
(6) Encounter a learning datum.

Datum consists of input and output, e.g. /k’at’a/ --> [k’ata].

In implementations of the algorithm, learning data are selected at random, and can be weighted for frequency.

(7) Generate an effective ranking and calculate the optimal candidate under that ranking.

<table>
<thead>
<tr>
<th>/k’at’a/</th>
<th>IDENT-IO (PLACE)</th>
<th>IDENT-IO (LARYNGEAL)</th>
<th>-LARYNGEAL SIMILARITY</th>
<th>Be IDENTICAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>k’ata</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>k’at’a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(8) If that candidate matched the learning datum, do nothing. Otherwise compare the real winner to the spurious winner, canceling any marks that they share.

<table>
<thead>
<tr>
<th>/k’at’a/</th>
<th>IDENT-IO (PLACE)</th>
<th>IDENT-IO (LARYNGEAL)</th>
<th>-LARYNGEAL SIMILARITY</th>
<th>Be IDENTICAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>k’ata</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>k’at’a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(9) Demote the constraints for which the winner has uncancelled marks, and promote the constraints for which the loser has uncancelled marks.

Demotion and promotion here mean adjusting the ranking value up or down by some small amount (the “plasticity”). Typically, the plasticity might start out at 2 and gradually decrease to 0.002.

(10) Repeat from (6) forever.

As the ranking values are adjusted, mistakes will become less and less frequent, and adjustments will happen rarely or never.

(The idea of maintaining a lifelong, though narrowed, ability to adjust one’s grammar is appealing: accents can drift when people move, for example.)
Thoughts on the GLA

(11) Robustness to errors
An isolated speech error’s effect is only as big as the plasticity at the time.

Before error (assuming initial MARKEDNESS >> FAITHFULNESS)

After hearing “fourth-floor dlorm room”

(12) Handling of variability
Imagine free variation between /kʼatʼa/ ? [kʼata] and /kʼatʼa/ ? [kʼatʼa]. Then, if LARSIM and Id[LAR] have the same ranking value, out of 100 encounters with this word, 25 will promote LARSIM and demote IDENT[LAR], and 25 will do the opposite (in 50 encounters, the learner will produce the right output and no learning will occur).

If no other data are relevant to these two constraints, they will continue to jitter around the same ranking value, causing the learner to reproduce the free variation of the environment.
(13) *Inherently ranked constraint families*

I don’t know if anyone’s looked at these.

We could adopt a convention that if C1 inherently dominates C2, if C1 is demoted C2 also is, and if C2 is promoted C1 also is.

(14) *Realism re. real acquisition*

GLA can do a good job of modeling actual acquisition: gradualness of learning, variability and overlap of stages, rapid initial improvement that levels off.


(15) *Frequency*

GLA predicts that frequently violated constraints get demote faster than infrequently violated constraints. This seems to match the facts in Boersma & Levelt, Curtin.

(16) *Subterranean constraints*

Sensitivity to frequencies means that “subterranean grammars” can be learned too. Subterranean grammar = ranking of constraints that are apparent in the statistics of the lexicon, but play no role in generating utterances.

Example: English OCP constraints (*slill, *sman). In the adult grammar, faithfulness along determines whether a root has m or p after s. But until Faithfulness climbs to the top of the grammar and errors cease, the learner keeps promoting and demoting irrelevant constraints according to their frequency of violation in the lexicon.

(17) *Apparent stages*

During a discussion at the 2001 SWOT, Mike Hammond pointed out that, in stochastic OT, steady movement of ranking values leads to the appearance of stable periods punctuated by short bursts of variation (does this happen in real acquisition?):
(18) **Power of the algorithm**

Can it be proven that any linear ranking is learnable with the GLA?

Not every pattern of variation is representable with stochastic OT, so a fortiori not every pattern of variation is learnable by the GLA. But is every representable pattern of variation learnable?

**Turkel’s Genetic Algorithm**

(19) **Genetic Algorithms**

GAs use the mechanisms of mutation and natural selection to solve problems. They’re used in engineering, artificial life, for evolving cellular automata…all over the place.

Example to try: http://ai.bpa.arizona.edu/~mramsey/ga.html

- Start with a set of solutions
- Allow the fittest ones to “mate” and mutate; kill the rest
- Repeat above step with “offspring” solutions

(20) **Rankings as chromosomes**

Each solution in the population is a linear ranking:

A>>B>>C>>D>>E>>F

(21) “Mating”: **One-point crossover adapted to constraint rankings:**

A>>B>>C>>D>>E>>F → A>>B>>C>>F>>E>>D
F>>E>>C>>A>>D>>B → F>>E>>C>>A>>B>>D

(22) **Mutation**

- Swap adjacent
  A>>B>>C>>D>>E>>F → A>>C>>B>>D>>E>>F

- Swap nonadjacent
  A>>B>>C>>D>>E>>F → A>>E>>C>>D>>B>>F
Reverse segment
A>>B>>C>>D>>E>>F ➔ A>>D>>C>>B>>E>>F

Rotate left
A>>B>>C>>D>>E>>F ➔ B>>C>>D>>E>>F>>A

Rotate right
A>>B>>C>>D>>E>>F ➔ F>>A>>B>>C>>D>>E

How it works
(23) Randomly generate a set of constraint rankings

\[
\text{ID[PLACE]>>*LARSIM>>BEIDENT>>ID[LAR]}
\]
\[
\text{ID[PLACE]>>*LARSIM>>ID[LAR]>>BEIDENT}
\]
\[
*\text{LARSIM}>>\text{ID[PLACE]>>BEIDENT}>>\text{ID[LAR]}
\]
\[
\text{BEIDENT}>>\text{ID[LAR]>>ID[PLACE]>>*LARSIM}
\]
\[
\text{ID[LAR]>>ID[PLACE]>>*LARSIM>>BEIDENT}
\]

(24) Assess the fitness of each ranking

Assume every constraint is binary. Then we can describe each candidate’s constraint violations as a binary number (“M-ranking”):

<table>
<thead>
<tr>
<th>/k’at’a/</th>
<th>-LARYNGEAL SIMILARITY</th>
<th>IDENT-IO (LARYNGEAL)</th>
<th>IDENT-IO (PLACE)</th>
<th>BE IDENTICAL</th>
<th>harmony</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>k’ata</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>0101</td>
</tr>
<tr>
<td>b</td>
<td>k’ata</td>
<td>*</td>
<td></td>
<td>*</td>
<td>1001</td>
</tr>
<tr>
<td>c</td>
<td>t’at’a</td>
<td>*</td>
<td></td>
<td>*</td>
<td>1010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>/t’ata/</th>
<th>-LARYNGEAL SIMILARITY</th>
<th>IDENT-IO (LARYNGEAL)</th>
<th>IDENT-IO (PLACE)</th>
<th>BE IDENTICAL</th>
<th>harmony</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>t’ata</td>
<td>*</td>
<td>*</td>
<td></td>
<td>1100</td>
</tr>
<tr>
<td>e</td>
<td>t’ata</td>
<td></td>
<td></td>
<td>*</td>
<td>0001</td>
</tr>
</tbody>
</table>

Low numbers represent greater harmony (zeros at the beginning = satisfaction of high-ranked constraints).

The fittest ranking is the one that assigns the lowest average number to the winning candidates

\[
\text{ID[PLACE]>>*LARSIM>>BEIDENT}>>\text{ID[LAR]}
\]
\[
\text{k’ata 0011} \quad \text{t’at’a 0101}
\]
\[
\text{ID[PLACE]>>*LARSIM}>>\text{ID[LAR]>>BEIDENT}
\]
\[
\text{k’ata 0011} \quad \text{t’at’a 0110}
\]
\[
*\text{LARSIM}>>\text{ID[PLACE]>>BEIDENT}>>\text{ID[LAR]}
\]
\[
\text{k’ata 0011} \quad \text{t’at’a 1001}
\]
\[
\text{BEIDENT}>>\text{ID[LAR]>>ID[PLACE]>>*LARSIM}
\]
\[
\text{k’ata 1100} \quad \text{t’at’a 0101}
\]
\[
\text{ID[LAR]>>ID[PLACE]>>*LARSIM>>BEIDENT}
\]
\[
\text{k’ata 1001} \quad \text{t’at’a 1010}
\]
Choose the fittest constraints, and allow them to mutate and combine

fittest constraints
ID[PLACE]>>*LARSIM>>BEIDENT>>ID[LAR]
ID[PLACE]>>*LARSIM>>ID[LAR]>>BEIDENT
ID[LAR]>>ID[PLACE]>>*LARSIM>>BEIDENT

mutations and combinations
ID[LAR]>>ID[PLACE]>>*LARSIM>>BEIDENT (crossover)
*LARSIM>>ID[LAR]>>BEIDENT>>ID[PLACE] (rotate left)

Repeat from (24) until grammar converges

How do we decide when to stop: when none of the mutations are better than the best ranking of the previous $n$ generations?

Plausibility
Boersma points out the problem of having to keep many hypothesized grammars in mind at once.
How do you choose which one to use in production?

It would be fun to write an implementation of this.

Next time
- Some issues in OT learning:
  - The initial state
  - Lexicon Optimization
  - Richness of the Base
- Learnability Theory in general

For next time
- Reading: Tesar & Smolensky ch. 5
- Optional reading: Jain et al. ch. 3
- Homework: play with GLA
  - Using either the same OT paper as in last week’s Constraint Demotion assignment or a different set of data, use OTSoft to run the Gradual Learning Algorithm. Turn in the final constraint ranking, the results of testing the grammar, and a graph of the ranking history. You should run the GLA at least twice, changing plasticity, noise, or initial ranking values. Discuss the differences, if any, between the results that you got on each run—can you tell what caused the differences?
  - If you really object to using Windows, you can instead download and use Praat, but it is not as user-friendly as OTSoft: [www.phon.hum.uva.nl/praat](http://www.phon.hum.uva.nl/praat)
Directions for running GLA in OTSoft

1. Create Excel file

- Create Excel file inputs outputs
- constraint names (full)
- constraint names (short)
- number of violations (blank = 0)

<table>
<thead>
<tr>
<th></th>
<th>ident[place]</th>
<th>Beldentical</th>
<th>ident[laryngeal]</th>
<th>*LaryngealSimilarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>k'ata k'ata</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k'ata</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>t'ata</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t'ata</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>t'ata</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If you want a constraint to be treated as a faithfulness constraint, its name must begin with “Id”, “Max”, “Dep”, or “Fai”. If you want a constraint to be treated as a markedness constraint, its name must begin with something else.

I’ll refer to this file as myfile.xls, but you should name it something more memorable.

2. Download OTSoft
http://www.linguistics.ucla.edu/people/hayes/otsoft/

3. Run the GLA
Double-click on otsoft.exe, then click “Work with different file” and find your Excel file. Choose the GLA button on the left side of the screen, and click on “Rank myfile.xls”. Play around with the options on the screen that appears, then click “Run GLA”.

4. View the results
Click “View results” back on the main screen. At the top you’ll see the final ranking values that the algorithm achieved, then the results of applying that ranking to all the inputs. If there weren’t enough learning trials, you’ll see some error.
If you want to cut and paste this information, look for a file called myfile.out, in the same folder as myfile.xls

5. View the ranking history
In the same folder as myfile.xls, you should see a file myfileHistory.xls. Open that file. You’ll see the constraint names across the top row; if they’re shifted over to the right, move them over so that they line up with the columns of numbers. Each row of numbers...
represents the ranking value of each constraint at one timestep. You’ll see that most of the ranking values change over time.

To make a graph of the ranking history, click in the top left cell, then click on the Chart Wizard button on the toolbar (or go to the Insert menu and select Chart…). Choose the chart type Line, and chart sub-type Line (the upper-left option). Click Finish.