Automatically Exploring Hypothesis Spaces

March 18, 2018

1 Automatically generating grammars with Bayesian program learning

1.1 A Probabilistic Framing

A goal of the grammar learner, whether it is a child, linguist, or computer program, is to infer the grammar that gave rise to a collection of utterances. One way of modeling this inference problem is to (1) place a prior distribution over grammars, for example, a distribution that puts more probability mass on shorter or simpler grammars; (2) equip each grammar with a *likelihood model* that specifies exactly how likely a grammar is to produce a given utterance; and then (3) use Bayes's rule to work backwards from the utterances to the grammar that was likely to produce them:

$$\mathbb{P}\left[\text{grammar}|\text{utterances}\right] \propto \underbrace{\mathbb{P}\left[\text{utterances}|\text{grammar}\right]}_{\text{Likelihood}} \times \underbrace{\mathbb{P}\left[\text{grammar}\right]}_{\text{Prior}} \tag{1}$$

We can think of the above equation as a recipe for scoring how well a grammar explains a collection of utterances. Once we have a prior and a likelihood, we know how to evaluate how well a grammar explains the data, modulo the assumptions implicit in the prior and likelihood. In this work we will use this probabilistic framing to explore spaces of grammars that an AGL learner could use to explain the pronunciations of words. Here the grammars will be SPE-style rewrite rules [1], commonly used in generative phonology, but at a high level this framing can apply to learning other aspects of grammar like semantics [2], syntax [3], and morphology [4].

SPE-style rewrite notation: Each SPE-style rewrite rule is a function that both inputs and outputs a sequence of phonemes. These rewrites are written as "input → output / left_right" which means that "input" gets rewritten to "output" whenever "left" is to the left and "right" is to the right. Some special symbols used in these rules are # to mean the start or end of the input; C/V to mean a consonant/vowel; σ to mean a syllable; and $\varnothing$ to mean the empty string. For example, the rule $\sigma \rightarrow \varnothing$ / #_C deletes a syllable at the start of a word if it is followed by a consonant. Rules can refer to sets of phonemes by writing down vectors of phonological features – for example, the rule [-son] → [+voice] rewrites segments that are -sonorant (e.g., k,d,v,ʃ,...) to their voiced counterparts (e.g., g,d,v,ʒ,...). Rules can also bind variables to phonemes or syllables, which are written using subscripts; for example, $C_i \rightarrow \varnothing$ / $\_C_i$ deletes a consonant if it is followed by the same consonant, and $\varnothing \rightarrow \sigma_i$ / $\_\sigma_i$ inserts a copy of a word initial syllable. The special subscript "0" means zero or more occurrences of the subscripted segment, so for example V → [+hi] / [+hi][$]_0$_ means that vowels become +hi whenever there exists a +hi phoneme anywhere to the left.

Prior & Likelihood: A simple and intuitive prior over grammars is one which penalizes longer grammars and favors parsimonious grammars, for example defining $\mathbb{P}\left[\text{grammar}\right] \propto \exp\left(-\#\text{ symbols in grammar}\right)$. An example of a likelihood model, and the one we use here, is just to count the number of extra bits or symbols needed to encode an utterance given the grammar[1].

Now in theory this probabilistic framing has now bought us a procedure for automatically discovering grammars that do a good job explaining the data. In practice, the space of all grammars is infinite and combinatorial, and so cashing out this framing requires a efficient procedure for searching the space of grammars and honing in on those that maximize Equation 1. A subfield of computer science, called **program synthesis**, has developed efficient techniques for solving seemingly intractable combinatorial search problems like those that arise in grammar learning. We use a program synthesis tool called Sketch [5] to search for grammars maximizing 1, which was previously applied to language learning problems in [6]. For more detail on these program synthesis techniques we refer the reader to [5]. This cashing out of the framing makes our approach an instance of **Bayesian Program Learning (BPL)**; see [7].

1.2 Modeling solutions to AGL problems

The fundamental hypothesis underlying AGL research is that artificial grammar learning must engage some shared resource with first language acquisition – and so we can gain insights into first language acquisition by instead studying AGL in controlled experiments. Our demonstration here is that a BPL model can explain how a learner can quickly (e.g., from very little data) infer a grammar represented as SPE–style rules. We collected grammars from AGL datasets [8, 9, 10] and presented our BPL learner with data drawn from these grammars (Table 1), finding that our model automatically discovers the grammars that underly these data sets.

1.3 Automatically exploring the space of alternative hypotheses

Just like in first language acquisition, these AGL learning setups introduce a trade-off between grammars that are likely under the prior (and are therefore small; "parsimony") and grammars that assign high likelihood to the data (and therefore closely "fit the data", having high likelihood). For example, a learner could infer a grammar that just memorized to the data (perfect fit but poor parsimony) or it

[1]This is equivalent to a minimum description length (MDL) interpretation

Grammar	Example input to learner	Inferred grammar	Natural language analogues
ABA Marcus et al. 1999.	wofewo lovilo fimufi	$\varnothing \rightarrow \sigma_i$ / $\#\_\sigma\sigma_i$	Reduplication (eg, Tagalog)
ABB Marcus et al. 1999.	wofefe lovivi fimumu	$\varnothing \rightarrow \sigma_i$ / $\sigma_i\_\#$	Reduplication (eg, Tagalog)
ABx	wofeka lovika fimuka	stem+x	concatenative morphology
AAx Gerken 2006.	wowoka loloka fifika	stem+x $\varnothing \rightarrow \sigma_i$ / $\#\_\sigma_i$	reduplication concatenative morphology
AxA Gerken 2006.	wokawo lokalo fikafi	x+stem $\varnothing \rightarrow \sigma_i$ / $\#\_\sigma\sigma_i$	Infixing Reduplication
Pig Latin	pɪg→ɪgpe latɪn→atɪle æsk→æske	$\varnothing \rightarrow \mathrm{C}_i$ / $\#\mathrm{C}_i[\ ]_0\_\#$ $\varnothing \rightarrow \mathrm{e}$ / $\_\#$ $\mathrm{C} \rightarrow \varnothing$ / $\#\_$	

Table 1: Using BPL to model artificial grammar learning. Learner given 5 examples.

could infer a grammar that can generate every possible word (parsimonious but a poor fit). Effective grammar learners must navigate this trade-off.

To analyze this trade-off, we calculated its shape in the form of a **pareto front** [11]. In our setting, a pareto front is the set of all grammars that are not worse than another grammar along the two competing axes of parsimony and fit to data. Intuitively, grammars on the pareto front are ones which an ideal Bayesian or MDL learner prefers, *independent* of how the learner decides to relatively weight the prior and likelihood. Figure 1 diagrams the Pareto fronts for two AGL experiments as the number of example words provided to the learner is varied. What these Pareto fronts show is (1) the set of grammars entertained by the learner, and (2) how the learner weighs these grammars against each other as measured by the prior (parsimony) and the likelihood (fit to the data). We believe that the Pareto front is a useful way of viewing the space of possible grammars, and understanding the different trade-offs between the grammars.

As an example of how the Pareto front offers a lens on the space of possible hypotheses, consider the upper left Pareto front in figure 1 (*aab, 1 example*). Here the learner has seen a single word, [vɛfefe]. Some grammars living on the Pareto frontier are:

- **A grammar that does nothing:** In the lower right-hand corner of 1 is the grammar labeled "surface=underlying", which just says that every word (a "surface pronunciation") is represented ("underlyingly") literally how it is pronounced. The observed word [vɛfefe] is represented as /vɛfefe/, which has 6 symbols, giving a fit to the data of -6.

- **A grammar that duplicates syllables:** Highlighted in dark red in 1 is a grammar with the single rule $\varnothing \rightarrow\ \sigma_i$ / $\_\sigma_i\#$ which inserts a copy of the last syllable syllable. This rule generates the word [vɛfefe] by starting with the stem (i.e., underlying form) /vɛfe/ (which has 4 symbols, fit to data of -4) and then applying the rule in the grammar, which copies the last syllable and makes [vɛfefe].

- **A grammar that duplicates syllables and appends morphemes:** Highlighted in pink in 1 are grammars that duplicate a syllable but also append or prepend extra morphemes. These correspond to generalizations where the learner believes that different parts of [vɛfefe] correspond to prefixes or suffixes in the language. For example, the grammar in the upper left corner incorporates the suffix /fe/ as well as the prefix /vɛ/, and explains the observation using an underlying form that is completely empty (fit to data of 0) – this grammar has memorized the observed data, and so maximally compresses it, but at the cost of having many symbols inside of the grammar (22 symbols, vs 6 symbols in the grammar that just duplicates a syllable).

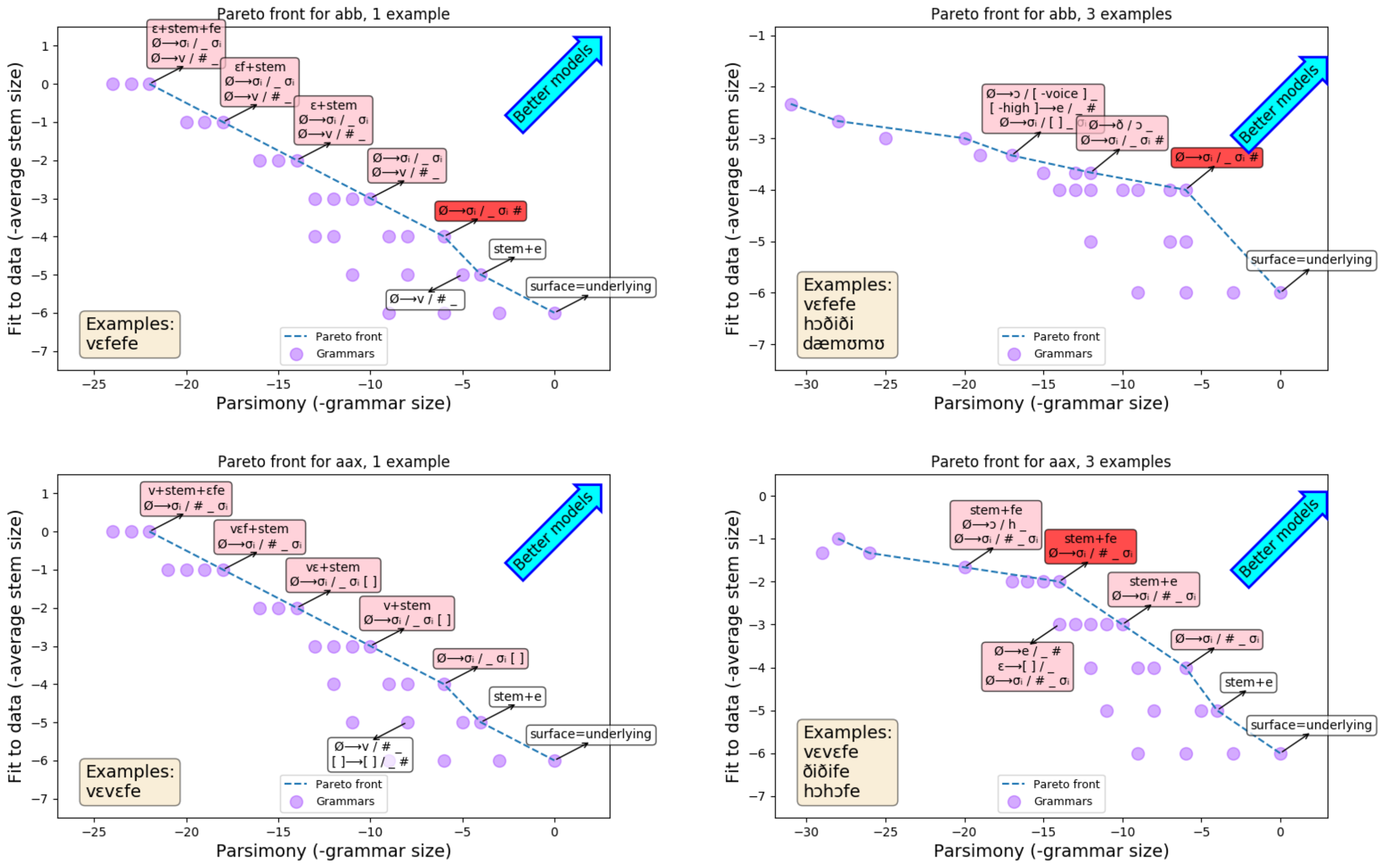

Figure 1: Pareto fronts for ABB (Marcus 1999; top two) & AAX (Gerken 2006; bottom two) learning problems fo either one example word (left column) or three example words (right column). Rightward on x-axis corresponds to more parsimonious grammars and upward on y-axis corresponds to grammars that best fit the data, so the best grammars live in the upper right corners of the graphs. Red shade: ground truth grammar. Pink shade: shares structure with ground truth grammar. White shade: incorrect generalizations. As the number of examples increases, the Pareto fronts develop a sharp kink around the ground truth grammar, which indicates a stronger preference for the correct grammar. With one example the kinks can still exists but are less pronounced.

1.4 Beyond artificial grammars: Applying BPL to natural language

A successful model of artificial grammar learning should also explain at least some aspects of first language acquisition. As a first step in this direction, we take the same model that we applied to these AGL learning problems and use it to solve phonology textbook exercises. Each of these textbook exercises is an inductive reasoning problem where the learner must explain surface pronunciations in terms of a latent generative model (a grammar).

These natural language results are still preliminary and ongoing. Figure 2 shows our model solving a representative textbook problem. We believe this extension to natural language is an important next step for models of artificial grammar learning, and helps deliver on the basic promises and presuppositions of AGL research.

Textbook Tibetan counting problem:

Number	Surface form
10	d^{ʒ}u
1	d^{ʒ}ig
11	d^{ʒ}ugd^{ʒ}ig
4	ʃi
40	ʃibd^{ʒ}u
9	gu
19	d^{ʒ}urgu
5	ŋa

Our model's solution:

Induced phonological rule: C → ∅ / #_C

(Reduce word initial consonant clusters)

Induced underlying representations:

1	gd^{ʒ}ig
4	bʃi
5	ŋa
9	rgu
10	bd^{ʒ}u

Figure 2: Example textbook phonology exercise (left, taken from [12]) and the solution our model discovers for it (right). To solve this problem the model must jointly infer unobserved underlying representations for each of the Tibetan numbers, along with a phonological rule that explains their surface pronunciations. The same code that calculated the Pareto fronts (Figure 1) produces this analysis of the Tibetan count system.

References

[1] N. Chomsky and M. Halle. *The sound pattern of English*. Studies in language. Harper & Row, 1968.

[2] Steven Thomas Piantadosi. *Learning and the language of thought*. PhD thesis, Massachusetts Institute of Technology, 2011.

[3] Amy Perfors, Joshua B Tenenbaum, and Terry Regier. The learnability of abstract syntactic principles. *Cognition*, 118(3):306–338, 2011.

[4] Timothy J. O'Donnell. *Productivity and Reuse in Language: A Theory of Linguistic Computation and Storage*. The MIT Press, 2015.

[5] Armando Solar Lezama. *Program Synthesis By Sketching*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2008.

[6] Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. Unsupervised learning by program synthesis. In *NIPS*.

[7] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[8] LouAnn Gerken. Infants use rational decision criteria for choosing among models of their input. *Cognition*, 115(2):362–366, 2010.

[9] Gary F Marcus, Sugumaran Vijayan, S Bandi Rao, and Peter M Vishton. Rule learning by seven-month-old infants. *Science*, 283(5398):77–80, 1999.

[10] Michael C Frank and Joshua B Tenenbaum. Three ideal observer models for rule learning in simple languages. *Cognition*, 120(3):360–371, 2011.

[11] Christopher A Mattson and Achille Messac. Pareto frontier based concept selection under uncertainty, with visualization. *Optimization and Engineering*, 6(1):85–115, 2005.

[12] David Odden. *Introducing Phonology*. Cambridge University Press, 2005. Cambridge Books Online.