Sampling for Bayesian Program Learning

Kevin Ellis, Armando Solar-Lezama, and Joshua B. Tenenbaum
Massachusetts Institute of Technology

Problem statement

Bayesian Program Learning: Learn programs from input/output examples, framed as Bayesian inference. Given a prior over programs, condition on examples. In this work we approximate the posterior by a set of samples.

- **Efficient in practice**: synthesizes non-trivial programs in minutes
- **Theoretical guarantees**: our algorithm, ProgramSample, generates iid samples from a distribution that provably approximates the true posterior

Sampling text edit programs:

Input	Output
"1/21/2001"	"01"

A sampled program	English interpretation
`substr(pos('0',-1),-1)`	"last 0 til end"
`const('01')`	"output 01"
`substr(-2,-1)`	"take last two"

Two Key Ingredients

How can we get both practical performance and theoretical guarantees for a problem that feels so obviously intractable? Combine two ideas:

- **Sketching:** The program's high-level structure is already given as a *sketch*, like a recursive grammar over expressions. Consider *finite* programs (bounded size, bounded runtime, bounded memory consumption), which can be modeled in a constraint solver: this does the heavy lifting of searching for programs. Here we use a SAT solver.
- **Sampling via random XOR constraints:** Once a program learning problem has been converted to SAT, we can sample programs (SAT solutions) by adding random XOR constraints to the SAT formula, an idea first introduced in Gomes et al 2006.

Problem Framing

- Sketch specifies a large but finite set of programs, S
- Program description length: $|x|$ for $x \in S$
- Prior over programs: $\propto 2^{-|x|}$
- Posterior over programs: call this $p(x)$, defined as $\propto 2^{-|x|}\mathbb{1}[x \text{ consistent with input/output examples}]$

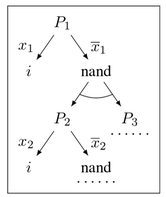
Our goal then is to sample from $p(\cdot)$.

Background: Program Synthesis by SAT Solving

```
Program ::= i
| nand(Program,Program)
```
(a) Sketch

```
Program(i = 0) = 1
```
(d) Specification

$x_1 = 0,\ x_2 = 1,\ x_3 = 1$
`Program = nand(i, i)`
(e) A constraint solution; $|x| = 3$ bits



(b) Program space

$i \Leftrightarrow 0 \wedge P_1 \Leftrightarrow 1$
$x_1 \Rightarrow (P_1 \Leftrightarrow i)$
$\overline{x}_1 \Rightarrow (P_1 \Leftrightarrow \overline{P_2 \wedge P_3})$
$x_2 \Rightarrow (P_2 \Leftrightarrow i)$
$\overline{x}_2 \Rightarrow (P_2 \Leftrightarrow \overline{P_4 \wedge P_5})$
$x_3 \Rightarrow (P_3 \Leftrightarrow i)$
$\cdots\cdots$
(c) Constraints for SAT solver

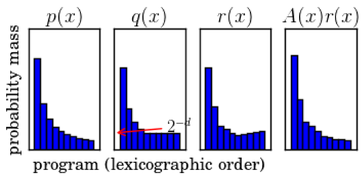
Figure 2: Synthesizing a program via sketching and constraint solving. `Typewriter font` refers to pieces of programs or sketches, while math font refers to pieces of a constraint satisfaction problem. The variable `i` is the program input.

Sampling by Random XOR Constraints

Embed X within a larger set E where sampling uniformly from E samples from $q(x) \propto 2^{-\min(|x|,d)}$, where d bounds the tilt of $q(\cdot)$:

$$E = \{(x, y) \ :\ x \in X,\ \bigwedge_{1 \le j \le d} |x| \ge j \Rightarrow y_j = 1\} \quad (1)$$

Sample approximately uniformly from E by fixing the parity of random sums (XOR's) of SAT variables. Use K such sums. # satisfying members of E is $\approx 2^{-K}|E|$. Accept sample w.p. $A(x) = \min(1, 2^{-|x|+d})$ to correct for $q(\cdot)$.



Summary. Approximate $p(x)$ by $q(x)$, embed in E, approximately uniform samples of E give $r(x)$, distribution of samples is $\propto A(x)r(x)$

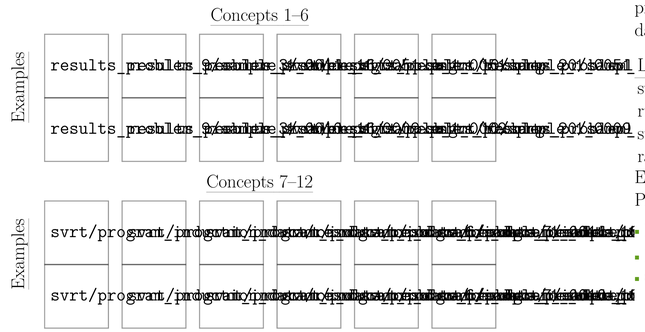
Pseudocode: Outer optimization loop

Minimize Eq. **??** with an SMT solver in an inner loop:

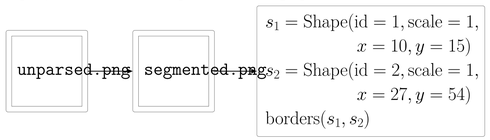
function UnsupervisedProgramSynthesis:
Input: Grammar $\mathcal{G}$, grammar start symbol P, denotation $[\![\cdot]\!]$, observations $\{x_i\}_{i=1}^N$, noise model, prior over program input
Output: Program f, N program inputs, description length ℓ
// I_i is an (unknown) program input the SMT solver will find
// Fresh variables unused in any existing formula
$I_1, I_2, \cdots, I_N =$ **FreshInputVariable()**
// Define hypothesis space and model program executions
// l_f = program description length, A = set of SMT constraints
$l_f, \{z_i\}_{i=1}^N, A \leftarrow$ **SMTEncoding**$(\mathcal{G}, [\![\cdot]\!], P, \{I_i\}_{i=1}^N)$
// Compute total description length ℓ
$\ell =$ **FreshRealVariable()**
$A \leftarrow A \cup \{\ell = l_f - \Sigma_i(\log P(x_i|z_i) + \log P(I_i))\}$
while A satisfiable according to SMT solver **do**
  $\sigma \leftarrow$ a satisfying solution to A
  $A \leftarrow A \cup \{\ell < \sigma[\ell]\}$
end while
let $f =$ unique program in $\mathcal{G}$ specified by σ
return $f, \{\sigma[I_i]\}_{i=1}^N, \sigma[\ell]$

Domain: Visual concepts

Humans can quickly learn many abstract visual concepts, often from few examples. Pairs of examples of twelve SVRT [**?**] concepts:



Our system learns SVRT concepts by synthesizing a program that can draw example images within that concept. Learning occurs not from pixel-level input, but from the output of an image parser:



Program representation:

- Program inputs: shapes, coordinates, distances, angles, scales
- Program output: Image parse
- Constraints on program space: control a turtle, but...
  - Restricted to alternatingly moving and drawning
  - No arithmetic on real variables
  - No rotation of shapes

Visual concepts: Classification performance

Given a test image t and a set of examples E_1 (resp. E_2) from class C_1 (resp. C_2), use decision rule $P_x(\{t\} \cup E_1)P_x(E_2) \gtrless_{C_2}^{C_1} P_x(E_1)P_x(\{t\} \cup E_2)$. Approximate marginals w/ their MDL: so 4 synthesis problems. Like humans we learn from $\approx$ 6 examples. Image features [**?**], ConvNet, parse features trained on 10000, 2000, 6 examples.

Domain: Linguistic rules

Children learn to inflect verbs without explicit stem/inflection pairs (eg, without supervision). Our system synthesizes some linguistic rules, represented as programs that transform a stem into its inflected forms. The learner's observed data consists of triples of ⟨lexeme, tense, word⟩. For example,

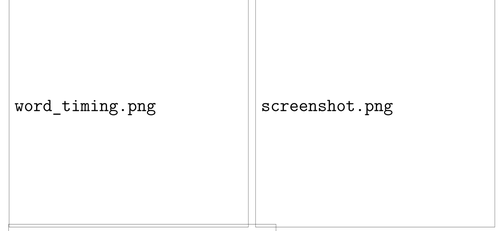
Lexeme	Present	Past	3rd Sing. Pres.	Past Part.	Prog.
style	staɪl	staɪld	staɪlz	staɪld	staɪlɪŋ
run	rʌn	ræn	rʌnz	rʌn	rʌnɪŋ
subscribe	səbskraɪb	səbskraɪbd	səbskraɪbz	səbskraɪbd	səbskraɪbɪŋ
rack	ræk	rækt	ræks	rækt	rækɪŋ

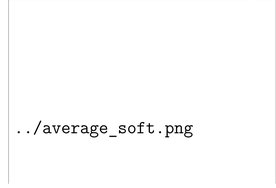
Each row of the above matrix is an observation (x_i's, or program outputs). Program representation:

- Program input: the underlying stem (sequence of phonemes)
- Program output: tuple of all inflections for a lexeme
- Constraints on program space: standard phonological primitives, but...
  - Has form: tuple of expressions, one for each tense.
  - Attend only to stem ending
  - Consider only suffixes

The noise model permits exceptions to learned rules (accommodating the irregulars). "Rules-plus-exceptions" model of the lexicon.

Linguistic rules: Model fitting





Synthesis time w/ an SMT solver grows quickly with increased training data, so we fit our models with Random Sample Consensus (RANSAC), recovering the above programs w/ a lexicon of 5000 lexemes.

References

Acknowledgements

We are grateful for discussions with Timothy O'Donnell, Brendan Lake, and Tejas Kulkarni. This material is based upon work supported by funding from NSF award SHF-1161775, from the Center for Minds, Brains and Machines (CBMM) funded by NSF STC award CCF-1231216, and from ARO MURI contract W911NF-08-1-0242.