Unsupervised Learning by Program Synthesis
Kevin Ellis, Armando Solar-Lezama, and Joshua B. Tenenbaum
Massachusetts Institute of Technology

Problem Statement

Discover latent structure in dataset, where that structure takes the form of a program. Examples of data→program:

- Visual concepts: gots(t, Rk). draw(shapes); move(t, f). draw(shapes, scale);
- Linguistic rules: (stem + /) raised [VOICED] (stem + /) else (stem + /)
- Multitask regression: (A [amplitude phase] X) [amplitude x sin(x + phase)]

In this work, we apply our methods to learning abstract visual concepts (eg, all in a line or horizontally symmetric) and to learning linguistic rules (eg, the rules that form verb inflections).

Why programs?

Why might a learner represent knowledge as a program, rather than the representations popular in much of machine learning, such as a large matrix of weights or a graphical model? Programs (probabilistic or deterministic) excel at representing knowledge that is (1) symbolic; (2) compositional; or (3) higher-order. We are motivated by the success of program induction in both AI domains, like semantic parsing and programing by demonstration, and in cognitive modeling, like intuitive theory learning [2].

The synthesis algorithm

Unsupervised program synthesis is a domain-general framework for defining domain-specific learning algorithms. For such domains, provide:

- The sketch: Define the program primitives, constrain the program space with a probabilistic context free grammar, giving P(ψ). Example:

  E = λz ∈ Z. E → φ

- N observations: (ψ1, ψ2, ..., ψn) [eg, words]
- Noise model Pψ(ψ) [eg, f(ξ)]

Minimize the description length:

\[ -\log P(\psi) = -\sum \left( \log P(\psi|\xi(\psi(\xi))) + \log P(\psi(\xi)) \right) \]  

(1)

Our solution

We extend two ideas from the program synthesis community to make search over programs tractable:

Sketching: Manually provide a sketch, or rough outline, of the program to be induced [3]. Our sketches are probabilistic context-free grammars.

Symbolic search: We automatically translate our sketches into Satisfiability Modulo Theories (SMT) problems. SMT problems are tractable in general, but often solved efficiently in practice, much like SAT problems.

Visual concepts: Classification performance

Humans can quickly learn many abstract visual concepts, often from few examples. Pairs of examples of two SNVT concepts [3]:

Program synthesis count:

- Concept 1.6: Almost 90% of the test data is classified correctly.

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 drawing
- No antecedents as well variables
- No rules of shapes

Visual concepts: Classification performance

Domain: Visual concepts

Children learn to infer words 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 (lemma, tense, word). For example,


run run run run run verb run
snore snore snore snore snore verb
shout shout shout shout shout verb

Approximate margins \( \theta \) for SMT synthesis problems. Like humans (we learn from 6 examples, Image features [1], ConvNet, parses features trained on 10,000, 2,000, 6 examples).

References


Acknowledgements

We are grateful for discussions with Timothee O’Donnell, Brandon Lake, and TokeKoldbaek. This manuscript is based upon research supported by funding from NSF award IIS-1327779, from the Center for Mind, Brain and Machines (CMBM) funded by NSF STCU award CCF- 1321212, and from ASU MIND under contract W3107Q-00-0-0041.

Linguistic rules: Mode fitting

Input: Grammar \( G \), grammar symbol start \( P_0 \), denotation \( [3] \) observations \( \epsilon \), noise model, prior over program input

Output: Program \( f \), \( N \) program inputs, description length \( \ell \), \( f \) is an (unobserved) program input the SMT solver will find

\( f \) fresh variables unused in any existing formula

\( I_0, I_1, I_2, I_3 \) = FreshInputVariable()

\( \ell \) = program description length, \( A \) set of SM constraints

\( A \), \( E \) = SMTEncoding\( [3] \) \( P \) \( I_0, I_1, I_2, I_3 \)

\( f \) = FreshRealVariable()

\( A \sim A + \mu \cdot (1 - \mu) \log P(\chi_0) + \mu \cdot \log P(\chi_1) \)

while \( f \) satisfies according to SMT solver do

end while

let \( f \) = unique program in \( G \) specified by \( \sigma \) return \( \sigma = [f(\chi), f(\chi_1)] \)

Pseudocode: Outer optimization loop

Initialization \( E \) is an input (\( \chi_0 \)), \( \epsilon \) is an output (\( \chi_1 \)). For example, \( \epsilon = \text{Shape}(x, \text{scale} = 1, y, x = 10, y = 15) \)

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 drawing
- No antecedents as well variables
- No rules of shapes

Synthesis time \( w \) on SMT solver grows quickly with increased training data, so we fit our models with Random Sample Consensus (RANSAC), recovering the above program \( w \) of a lesson of 5000 lemmas.