Inducing Domain Specific Languages for Bayesian Program Learning

Anonymous Authors [1]

Abstract

This document provides a basic paper template and submission guidelines. Abstracts must be a single paragraph, ideally between 4–6 sentences long. Gross violations will trigger corrections at the camera-ready phase.

1. Introduction

Imagine an agent faced with a suite of new problems totally different from anything it has seen before. It has at its disposal a basic set of primitives it can compose to build solutions to these problems, but it is no idea what kinds of primitives are appropriate for which problems nor does it know the higher-level domain-specific language in which solutions are best expressed.

Our algorithm accomplishes the following:

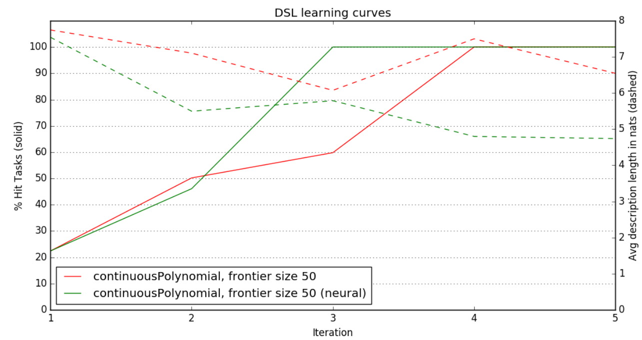
- Learns from relatively small amounts of data and with weak supervision. We do not use ground truth programs – weak supervision. Our DSL learner does not need huge amounts of data – tens of examples suffice.
- Jointly infers a *generative model* along with a *recognition model*. The *generative model* is a probabilistic context-sensitive grammar over programs, and includes a DSL. The *recognition model* is a neural network that guides the agents use of the DSL.

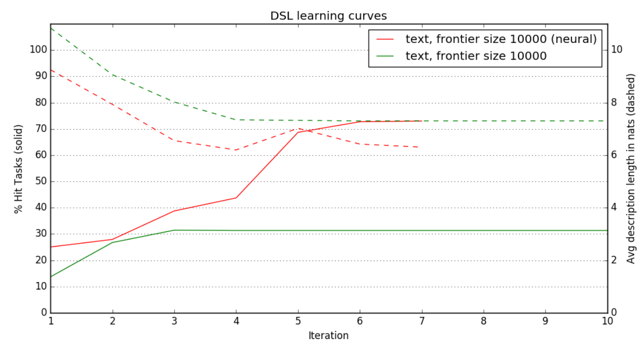
The generative model and the recognition model bootstrap off of each other:

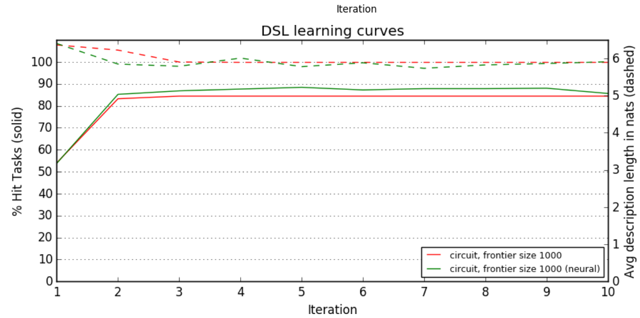
- The recognition model works by upweighting the probability of program components likely to be useful for a given problem. By learning a DSL, the recognition model gets more power because it can upweight new more powerful primitives that are better suited for the domain.

[1] Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.







- The generative model (DSL) is learned from programs that the agent has found so far. Because the recognition model speeds up search, it generates more training data for the generative model.

2. Experiments

3. Model

4. Implementation

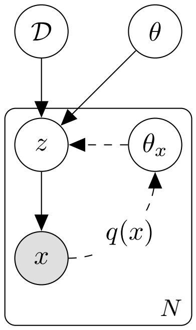


Figure 1: DSL $\mathcal{D}$ generates programs z by sampling DSL primitives with probabilities θ (Algorithm 1). We observe program outputs x. A neural network $q(\cdot)$ called the *recognition model* regresses from program outputs to a distribution over programs ($\theta_x = q(x)$). Solid arrows correspond to the top-down generative model. Dashed arrows correspond to the bottom-up recognition model.

5. Estimating the grammar parameters

I justify this estimator by proving that it maximizes a lower bound on the log likelihood of the data. Writing L for the log likelihood, θ for the parameters of the grammar, N for the number of random choices, A to range over the alternative choices for a random variable, $c(x)$ to mean the number of times that primitive x was used, and $a(x) = \sum_A \mathbb{1}[x \in A]$ to mean the number of times that primitive x could have been used:

$$L = \sum_x c(x) \log \theta_x - \sum_A \log \sum_{x \in A} \theta_x \tag{1}$$

$$= \sum_x c(x) \log \theta_x - N \mathbb{E}_A \log \sum_{x \in A} \theta_x \tag{2}$$

$$\geq \sum_x c(x) \log \theta_x - N \log \mathbb{E}_A \sum_{x \in A} \theta_x, \text{ Jensen's inequality} \tag{3}$$

$$= \sum_x c(x) \log \theta_x - N \log \frac{1}{N} \sum_A \sum_x \mathbb{1}[x \in A] \theta_x \tag{4}$$

$$\stackrel{+}{=} \sum_x c(x) \log \theta_x - N \log \sum_x a(x) \theta_x. \tag{5}$$

Differentiate with respect to θ_x and set to zero:

$$\frac{c(x)}{\theta_x} = N \frac{a(x)}{\sum_y a(y) \theta_y} \tag{6}$$

This equality holds if $\theta_x = c(x)/a(x)$:

$$\frac{c(x)}{\theta_x} = a(x). \tag{7}$$

$$N \frac{a(x)}{\sum_y a(y) \theta_y} = N \frac{a(x)}{\sum_y c(y)} = N \frac{a(x)}{N} = a(x). \tag{8}$$

Algorithm 1 Generative model over programs

function sample($\mathcal{D}, \theta, \mathcal{E}, \tau$):
Input: DSL $\mathcal{D}$, weight vector θ, environment $\mathcal{E}$, type τ
Output: a program whose type unifies with τ
if $\tau = \alpha \to \beta$ **then**
  var $\leftarrow$ an unused variable name
  body $\sim$ sample($\mathcal{D}, \theta$, [var : α] $+ \mathcal{E}, \beta$)
  return λvar. body
end if

$$\text{primitives} \leftarrow \{p | p : \alpha \to \cdots \to \beta \in \mathcal{D} \cup \mathcal{E} \text{ if canUnify}(\tau, \beta)\}$$

Sample $e \sim$ primitives, w.p. $\propto \theta_e$ if $e \in \mathcal{D}$ and w.p. $\propto \frac{\theta_{var}}{|\text{variables}|}$ if $e \in \mathcal{E}$
Let $e : \alpha_1 \to \alpha_2 \to \cdots \to \alpha_K \to \beta$. Unify τ with β.
for $k = 1$ **to** K **do**
  $a_k \sim$ sample($\mathcal{D}, \theta, \mathcal{E}, \alpha_k$)
end for
return $e(a_1, a_2, \cdots, a_K)$

If this equality holds then $\theta_x \propto c(x)/a(x)$:

$$\theta_x = \frac{c(x)}{a(x)} \times \underbrace{\frac{\sum_y a(y) \theta_y}{N}}_{\text{Independent of } x}. \tag{9}$$

Now what we are actually after is the parameters that maximize the joint log probability of the data+parameters, which I will write J:

$$J = L + \log \mathrm{D}(\theta | \alpha) \tag{10}$$

$$\stackrel{+}{\geq} \sum_x c(x) \log \theta_x - N \log \sum_x a(x) \theta_x + \sum_x (\alpha_x - 1) \log \theta_x \tag{11}$$

$$= \sum_x (c(x) + \alpha_x - 1) \log \theta_x - N \log \sum_x a(x) \theta_x \tag{12}$$

So you add the pseudocounts to the *counts* ($c(x)$), but not to the *possible counts* ($a(x)$).

Algorithm 2 DSL Learner

Input: Initial DSL $\mathcal{D}$, set of tasks X, iterations I
Hyperparameters: Frontier size F
Output: DSL $\mathcal{D}$, weight vector θ, bottom-up recognition model $q(\cdot)$
Initialize $\mathcal{D}_0 \leftarrow \mathcal{D}$, $\theta_0 \leftarrow$ uniform, $q_0(\cdot) = \theta_0$
for $i = 1$ **to** I **do**
  for $x : \tau \in X$ **do**
    $\mathcal{F}_x \leftarrow \{z | z \in \text{enumerate}(\mathcal{D}_{i-1}, q_{i-1}(x), F) \cup \text{enumerate}(\mathcal{D}_{i-1}, \theta_{i-1}, F) \text{ if } \mathbb{P}[x|z] > 0\}$
  end for
  $\mathcal{D}_i, \theta_i \leftarrow \text{induceGrammar}(\{\mathcal{F}_x\}_{x \in X})$
  Define $Q_x(z) \propto \begin{cases} \mathbb{P}[x|z]\mathbb{P}[z|\mathcal{D}_i, \theta_i] & x \in \mathcal{F}_x \\ 0 & x \notin \mathcal{F}_x \end{cases}$
  $q_i \leftarrow \arg\min_q \sum_{x \in X} \text{KL}(Q_x(\cdot) || \mathbb{P}[\cdot|\mathcal{D}_i, q(x)])$
end for
return $\mathcal{D}^I, \theta^I, q^I$

Algorithm 3 Grammar Induction Algorithm

Input: Set of frontiers $\{\mathcal{F}_x\}$
Hyperparameters: Pseudocounts α, regularization parameter λ, AIC coefficient a
Output: DSL $\mathcal{D}$, weight vector θ
Define $\log \mathbb{P}[\mathcal{D}] \stackrel{+}{=} -\lambda \sum_{p \in \mathcal{D}} \text{size}(p)$
Define $L(\mathcal{D}, \theta) = \prod_x \sum_{z \in \mathcal{F}_x} \mathbb{P}[z|\mathcal{D}, \theta]$
Define $\theta^*(\mathcal{D}) = \arg\max_\theta \text{Dir}(\theta|\alpha) L(\mathcal{D}, \theta)$
Define $\text{score}(\mathcal{D}) = \log \mathbb{P}[\mathcal{D}] + L(\mathcal{D}, \theta^*) - a|\mathcal{D}|$
$\mathcal{D} \leftarrow$ every primitive in $\{\mathcal{F}_x\}$
while true **do**
  N $\leftarrow \{\mathcal{D} \cup \{s\} | x \in X, z \in \mathcal{F}_x, s \text{ a subtree of } z\}$
  $\mathcal{D}' \leftarrow \arg\max_{\mathcal{D}' \in N} \text{score}(\mathcal{D}')$
  if $\text{score}(\mathcal{D}') > \text{score}(\mathcal{D})$ **then**
    $\mathcal{D} \leftarrow \mathcal{D}'$
  else
    return $\mathcal{D}, \theta^*(\mathcal{D})$
  end if
end while