PROGRAM SYNTHESIS WITH NOISE

V. Raychev, Learning from Large Codebases, 2016

Anna Sokolova

January 24, 2018

Programming by example (PBE)

The user provides a number of examples the program should satisfy (the desired output for a given input) instead of directly providing a program.

Key problem:
Most of the PBE techniques can not adequately deal with incorrect examples as they attempt to satisfy *all* given examples, thus overfitting to the data. So, if the user makes a mistake while providing the examples, wrong program is produced.

Noise free synthesis

Counter-example guided inductive synthesis (CEGIS)):
A small set of examples $d \subseteq \mathcal{D}$ is selected, s.t. synthesizing on d generates the desired program.

Data: A large dataset $\mathcal{D}$ of examples
Result: Program p
initialization: random $d \subset \mathcal{D}$;
Generate program p for d;
while *a program p satisfies not all examples in $\mathcal{D}$* **do**
  Add examples that are not satisfied by the last generated program p to d;
  Generate new candidate program p for current set d.
end

Algorithm 1: Noise-free Synthesis

Quantifying noise

Two cases for quantifying the noise in the dataset:

- Bounded noise. The optimality guarantees on the learned program are provided.
- Arbitrary noise. The learning algorithm is approached with a fast, scalable algorithm for performing approximate empirical risk minimization (ERM).

Problem formulation

Let $\mathcal{D}$ be a set of examples and $\mathbb{P}$ be the set of all possible programs. The dataset $\mathcal{D}$ may contain errors, i.e. examples which the program should not satisfy.

The **objective** is to discover a program in $\mathbb{P}$ which satisfies the examples in $\mathcal{D}$.

Let $\mathcal{P}(\mathcal{D})$ be the set of all the finite subsets of dataset $\mathcal{D}$, $r : \mathcal{P}(\mathcal{D}) \times \mathbb{P} \to \mathbb{R}$ be a cost (or risk) function that given a dataset and a program, returns a non-negative real value that determines the inferiority of the program on the dataset.

Formal problem statement:

The synthesis problem is to find the program with the lowest cost on the entire dataset:

$$p_{best} = \arg\min_{p \in \mathbb{P}} r\left(\mathcal{D}, p\right)$$

Challenges:

- Dataset $\mathcal{D}$ may be prohibitively large, or simply infinite and thus, p_{best} can not be learned directly.
- To show that a program p is optimal, we should rank it with respect to *all* possible programs in $\mathbb{P}$.

Thus, the problem is mitigated.

Relaxed problem statement:
The synthesis problem is to find the *satisfactory* program $p^{\approx best}$ with the cost close to the cost of the best program p_{best} or is better than a given noise bound.

$$r(\mathcal{D}, p^{\approx best}) < r\left(\mathcal{D}, p_{best}\right) + \varepsilon$$

Iterative synthesis algorithm

The algorithm consists of two separated components: a *program generator* and *dataset sampler*, linked together in a feedback loop.

- **Program generator** is a function $gen : \mathcal{P}(\mathcal{D}) \to \mathbb{P}$ defined as follows:

$$gen(d) = \arg\min_{p \in \mathbb{P}} r(d, p).$$

  Since the invocations to $gen(d)$ are assumed to be expensive, the dataset d should be as small as possible.

- **Dataset sampler** is a function $ds : \mathcal{P}(\mathbb{P}) \times \mathbb{N} \to \mathcal{P}(\mathcal{D})$:

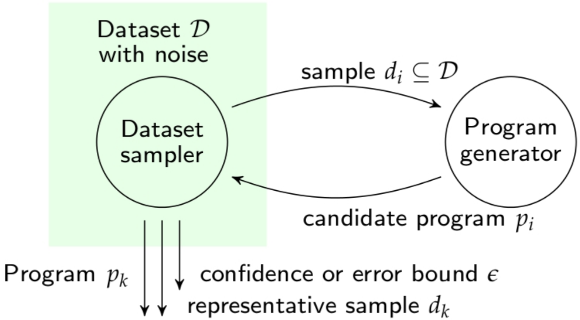
$$ds(progs, n) = d' \quad \text{with} \quad |d'| \geqslant n$$

Input: Dataset $\mathcal{D}$, initial (e.g. random) dataset $\emptyset \subset d_1 \subseteq \mathcal{D}$
Output: Program p

begin
    $progs \leftarrow \emptyset$
    $i \leftarrow 0$
    repeat
        $i \leftarrow i + 1$
        // *Dataset sampling* step
        if $i > 1$ **then**
            $d_i \leftarrow ds(progs, |d_{i-1}| + 1)$
        end
        // *Program generation* step
        $p_i \leftarrow gen(d_i)$
        if $found\_program(p_i)$ **then**
            return p_i
        end
        $progs \leftarrow progs \cup \{p_i\}$
    until $d_i = \mathcal{D}$;
    return "No such program exists"
end

Algorithm 2: Program Synthesis with Noise

Synthesis with noise



Reduction of search space

The size of dataset d increases at every step, while the goal is to discover a good program using only a small dataset d. So, it is very important to carefully pick small datasets the trim the space of possible programs.

- **Noise-free search space pruning.** The program generated at step i is different from the previously generated programs:

$$gen(d_i) \notin \{p_j\}_{j=1}^{i-1}$$

- **Pruning search space with noise.** A generated program p is kept in the candidate program space if it is within some distance ε of p_{best}:

$$r(\mathcal{D}, p) \leqslant r(\mathcal{D}, p_{best}) + \varepsilon.$$

Space pruning

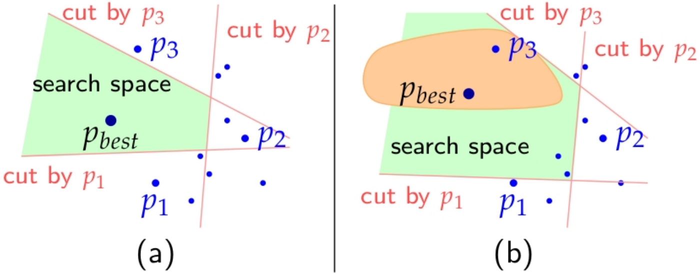


Figure: Search space pruning without noise (a) and with noise (b).

A hard dataset sampler is a function ds^H such that for $Q \subseteq \mathbb{P}$, $d' = ds^H(Q, min\_size)$, it holds that $\forall p \in Q \; r(\mathcal{D}, p) \leqslant r(d', p)$ and $|d'| \geqslant min\_size$.
Hard dataset sampler always exists (we can take $d' = \mathcal{D}$).

This definition generalizes the concept of providing more examples in CEGIS (noise-free synthesis). If there is an unsatisfied example x in $\mathcal{D}$, it is included in d'. Since SEGIS does not handle noise, $r(d, p)$ returns 0 if the program p satisfies all examples in d and 1, otherwise. Hence, it is a hard dataset sampler.

Theorem

Let $Q = \{p_1, \ldots, p_{i-1}\}$ be the set of programs generated up to iteration i of Algorithm 2, where the dataset sampler ds is hard. If $\varepsilon \geqslant r(d_i, p_{best}) - r(\mathcal{D}, p_{best})$, then $p_i = gen(d_i) \notin Q'$ where:

$$Q' = \{p \in Q | r(\mathcal{D}, p) > r(\mathcal{D}, p_{best}) + \varepsilon\}$$

Theorem

Let $Q = \{p_1, \ldots, p_{i-1}\}$ be the set of programs generated up to iteration i of Algorithm 2, where the dataset sampler ds is hard. If $\varepsilon \geqslant r(d_i, p_{best}) - r(\mathcal{D}, p_{best})$, then $p_i = gen(d_i) \notin Q'$ where:

$$Q' = \{p \in Q | r(\mathcal{D}, p) > r(\mathcal{D}, p_{best}) + \varepsilon\}$$

Proof.

Let $p \in Q'$. Then $r(\mathcal{D}, p) > r(\mathcal{D}, p_{best}) + \varepsilon$.
From the definition of ε: $\varepsilon + r(\mathcal{D}, p_{best}) \geqslant r(d_i, p_{best})$, hence, $r(\mathcal{D}, p) > r(d_i, p_{best})$.
$d_i = ds^H(Q, \_) \quad \Rightarrow r(d_i, p) \geqslant r(\mathcal{D}, p)$.
From the last two statements follows that $r(d_i, p) > r(d_i, p_{best})$.
But $p_i = gen(d_i) = \arg\min_{p' \in \mathbb{P}} r(d_i, p_{best})$, thus, $p_i \neq p$ and $p_i \notin Q'$. $\square$

Representative Dataset Sampler

Representativeness measure

$$repr(Q, \mathcal{D}, d) = \max_{p \in Q} |r(\mathcal{D}, p) - r(d, p)|$$

Representative dataset sampler

$$ds^R(Q, size) = \arg \min_{d \subseteq \mathcal{D}, |d| = size} repr(Q, \mathcal{D}, d)$$

If $d' = ds^R(Q, size)$ is such that $repr(Q, \mathcal{D}, d') = 0$ then the produced dataset is *perfectly representative*. In this case ds^R is also a hard dataset sampler, because $\forall p \in Q \; r(\mathcal{D}, p) = r(d', p)$.
Intuition of the requirement: If the example is incorrect, it will likely behave similarly on all programs. Thus, if we find small ε on several already explored programs, a similar bound may be true for all programs and for p_{best}.

Theorem

Let $Q = \{p_1, \ldots, p_{i-1}\}$ *be the set of programs generated up to iteration* i *of Algorithm 2. Let* $p_k = \arg\min_{p' \in Q} r(\mathcal{D}, p')$ *be the best program explored so far. By definition,* $p_k \in Q$*. Let* $\delta = repr(Q, \mathcal{D}, d_i)$ *be the representativeness measure of* d_i *and* d_i *is obtained from a representative dataset sampler as* $d_i \leftarrow ds^R(Q, size)$*. Then* $p_i = gen(d_i) \notin Q'$ *where*

$$Q' = \{p \in Q | r(\mathcal{D}, p) > r(\mathcal{D}, p_k) + 2\delta\}$$

Note, that the set Q' has the same shape as in Theorem for hard sampler except that we consider p_k(best program so far) instead of p_{best}(best program globally), and 2δ instead of ε.

Theorem

Let $Q = \{p_1, \ldots, p_{i-1}\}$, $p_k = \arg\min_{p' \in Q} r(\mathcal{D}, p')$, $\delta = repr(Q, \mathcal{D}, d_i)$ *and* $d_i = ds^R(Q, size)$. *Then* $p_i = gen(d_i) \notin Q'$ *where:* $Q' = \{p \in Q | r(\mathcal{D}, p) > r(\mathcal{D}, p_k) + 2\delta\}$

Proof.

Let $p \in Q'$. Since $Q' \subseteq Q$ and $\delta = repr(Q, \mathcal{D}, d_i)$, we get $|r(d_i, p) - r(\mathcal{D}, p)| \leqslant \delta$, and hence, $r(\mathcal{D}, p) \leqslant r(d_i, p) + \delta$.
Similarly, $p_k \in Q$, thus, $r(d_i, p_k) \leqslant r(\mathcal{D}, p_k) + \delta$.
$p_k = \arg\min_{p' \in Q} r(\mathcal{D}, p')$ and $p \in Q \quad \Rightarrow \quad r(\mathcal{D}, p_k) < r(\mathcal{D}, p)$.
Then, we obtain that:

$$r(d_i, p_k) \leqslant r(\mathcal{D}, p_k) + \delta < r(\mathcal{D}, p) + \delta \leqslant r(d_i, p) + 2\delta$$

Finally, $r(d_i, p_k) < r(d_i, p) \;\Rightarrow\; p \neq \arg\min_{p' \in \mathbb{P}} r(d_i, p')$ and as a result $p \neq p_i = gen(d_i)$. $\square$

Cost Functions

We have defined the synthesis problem to minimize the cost of a program p on a dataset d. Concrete cost functions:

- $num\_errors(d, p)$ returns the number of errors a program p does on a dataset of examples d.
- $error\_rate(d, p) = \frac{num\_errors(d,p)}{|d|}$ is the fraction of the examples with an error.
- Other measures weight the errors done by the program p on the dataset d according to their kind (e.g., entropy is one possible such measure.)

Regularization

For a cost metric r, its regularized version

$$r_{reg}(d, p) = r(d, p) + \lambda \cdot \Omega(p).$$

The regularizer $\Omega(p)$ aims to penalize programs which are too complex and prevent overfitting to the data.
The regularizer does not have access to dataset d, but only to given program p.