Inductive synthesis for variable RL

Monday, 9 November 2020 23:10

- Rather than enforcing safety by examining and altering the structure of complex NN implementations → use black-box methods to synthesizes deterministic programs, simpler, more interpretable approximations of NN.
- Forms the problem of NN verification in terms of: ① counterexample and ② syntax-guided inductive synthesis procedure.
- The procedure searches for ③ a deterministic program and ④ an inductive invariant over an infinite state transition system that represents a specification of an applicantion's control logic.
- The synthesis procedure treats the NN as an oracle extracting a deterministic program P intended to approximate the policy actions implemented by the network
  P also serves as a safety shield that operates in tandem with the network
- Pros: generalizes to infinite state systems with a continuous state space.
- A verification toolchain for ensuring that the control policies learned by RL-based neural network are safe.
- Verification Procedure:
  - To verify NN control policy π_w w.r.t environment context C we have to synthesize a deterministic policy program P.
  - P should: 1. broadly resembles its neural oracle
    2. satisfies a desired safety property when deployed in C.
  - Safety proof of $C[P]$ is easier than of $C[\pi_w]$

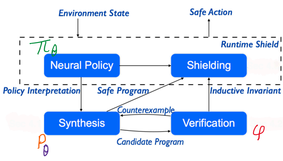

Figure 2. The Framework of Our Approach.

1. Synthesis:

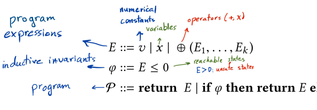

Figure 5. Syntax of the Policy Programming Language.

- The sketch $P[\theta]$ describes the shape of target policy using the grammar.
  $P[\theta]$: θ unknown variables, P_θ : θ known variables (P_θ is the synthesized program).
  ex: $P[\theta](x) :=$ return $\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n + \theta_{n+1}$
- the goal of the algorithm is to find θ that maximize the likelihood that P_θ resembles the neural oracle π_w, while still being safe w.r.t. C:
  $$\theta^* = \underset{\theta \in \mathbb{R}^{n+1}}{\arg\max}\ d(\pi_w, P_\theta, C)$$
- we approximate d by randomly sampling a set of trajectories C that are encountered by P_θ in the environment transition system $C[P_\theta]$: $d(\pi_w, P_\theta, C) = \sum_{h \in C} d(\pi_w, P_\theta, h)$
  for each trajectory h: $d(\pi_w, P_\theta, h) = \sum_{t=0}^{T} \begin{cases} -\|P_\theta(s_t) - \pi_w(s_t)\| & : s_t \notin S_0 \\ -Max & : s_t \in S_0 \end{cases}$

Algorithm 1: Synthesize (π_w, $\mathcal{P}[\theta]$, $C[\cdot]$)
(π_w: NN oracle; $\mathcal{P}[\theta]$: program; $C[\cdot]$: rollout)

```
θ ← 0;
do
    Sample δ from a zero mean Gaussian vector;
    Sample a set of trajectories C1 using C[P_{θ+νδ}];
    Sample a set of trajectories C2 using C[P_{θ−νδ}];
    θ ← θ + α[(d(π_w, P_{θ+νδ}, C1) − d(π_w, P_{θ−νδ}, C2)) / ν] δ;
while θ is not converged;
return P_θ
```

δ: Gaussian noise; ν: small real number; perturbed policy program

2. Verification:

Verification algorithm learns an inductive variant φ over the transition relation $\tau_t[P]$ formally proving that all possible system behaviors are encapsulated in φ and φ has to be disjoint with all unsafe states S_u.
by defining a function $E: \mathbb{R}^n \to \mathbb{R}$ that serves as a barrier between reachable states $E \leq 0$ and unsafe states $E > 0$

$\varphi[c] := E[c](X) \leq 0$; c parameters to be learned

E is found using SMT-solvers: $E[c](X) = \sum_{i=0}^{p} c_i b_i(X)$ (one term)

- The algorithm Counter Example-Guided Inductive Synthesis (CEGIS):

Algorithm 2: CEGIS (π_w, $\mathcal{P}[\theta]$, $C[\cdot]$)

```
1  policies ← ∅;
2  covers ← ∅;
3  while C.S0 ⊈ covers do                          (check by a SMT-solver)
4      search s0 such that s0 ∈ C.S0 ∧ s0 ∉ covers;   (sample counter-example initial state s0)
5      r* ← Diameter(C.S0);
6      do
7          φ_bound ← {s | s ∈ {s0 − r*, s0 + r*}};
8          C̃ ← C where C̃.S0 = (C.S0 ∩ φ_bound);      (safe policy)
9          θ ← Synthesize(π_w, P[θ], C̃[·]);          (reduce the initial state space)
10         φ ← Verify(C̃[P_θ]);
11         if φ is False then
12             r* ← r*/2;                            (shrink the radius)
13         else                                      (holds)
14             covers ← covers ∪ {s | φ(s)};
15             policies ← policies ∪ (P_θ, φ);
16             break;
17     while True;
18 end
19 return policies
```

the algorithm is sound but ① it may not terminate as r^* can become small
② no restriction on the size of potential counter example.

3. Sheilding:

Problem: the network may exhibit behaviors not captured by the simpler deterministic program.

Solution: using the synthesized policy program and its inductive invariant as a shield.

Algorithm 3: Shield (s, $C[\pi_w]$, $\mathcal{P}$, φ)

```
1 Predict s' such that (s, s') ∈ C[π_w].τ_t(s);   → predict the next state
2 if φ(s') then return π_w(s);
3 else return P(s);                               → shield to ensure safety
```

(P(s) is proved safe in Algorithm 2)

Figure 4. The Framework of Neural Network Shielding.

- Tools used: deep policy gradient for RL training
  Z3 SMT-solver to check convergence of CEGIS loop.
  Mosek constraint solver to generate inductive invariants of synthesized program from a sketch.
- Synthesizing the shield is better (faster) than retraining NN for a new environment.