Programmatically interpretable RL

Wednesday, 11 November 2020 22:48

- PIRL: is based on the idea of learning policies that are represented in a human-readable language.
- The problem: traditional RL + a (policy) sketch that syntactically defines a set of programmatic policies in PIRL language
- key benefits; 1. Interpretability.
  2. Implicitly encode the learner's inductive bias, which will be used for generalization.
  3. Effective pruning of undesired policies, to make the search for a good policy more efficient.
  4. Allow to use symbolic program verification techniques.
- key challenge: the space of policies permitted in an instance can be vast and non-smooth; making optimization extremely challenging.
- Solution: Neurally Directed Program Synthesis (NDPS) uses DRL to train NN → use the NN to direct a local search over programmatic policies then; interesting inputs → minimization the distance between ① the program output and ② the output of the oracle (NN policy).
- Partially observable Markov decision process (POMDP): $M = \{S, A, O, T, Z(\cdot|s), r, Init, \gamma\}$
  $Z(o|s)$ - the probability that the agent makes an observation o at state s.
  $Init$ - the initial distribution over environment states.
  A history $h = o_0, a_0, o_1, a_1, \dots, a_{k-1}, o_k$
  $\mathcal{H}_M$ a set of histories.
- A Programmatic language for policies:
  - atoms: observations and actions as well as auxiliary integers and reals generated during computation.
  - Two types of data: ① atoms and ② sequences of atoms (e.g. histories).

$$E ::= c \mid x \mid \oplus(E_1, \dots, E_k) \mid \textbf{peek}(x, i) \mid \textbf{fold}((\lambda x_1, x_2.\ E_1), \alpha)$$

(constants; expression; basic operators; observation from the history; combination of sequences; input)

most recent observation $peek(x, -1)$

$fold(f, [e_1, \dots, e_k], e) = f(e_k, f(e_{k-1}, \dots, f(e_1, e)))$

the system identifies a set of expressions whose ① inputs are histories ② outputs are actions

this expressions are known as programs (programmatic policies).

- sketch: is a grammar of expressions over atoms and sequences of atoms, places restrictions on the kind of basic operators that will be considered during policy search.
  the set of programs permitted by a sketch S is denoted $[\![S]\!]$
- The PIRL problem: find a program $e^* \in [\![S]\!]$ with optimal reward

$$e^* = \arg\max_{e \in [\![S]\!]} R(e)$$

example; PID controller.

sketch:

$$\begin{aligned}
P &::= \textbf{peek}((\epsilon - h_i), -1) \\
I &::= \textbf{fold}(+, \epsilon - h_i) \\
D &::= \textbf{peek}(h_i, -2) - \textbf{peek}(h_i, -1) \\
C &::= c_1 * P + c_2 * I + c_3 * D \\
B &::= c_0 + c_1 * \textbf{peek}(h_1, -1) + \dots > 0 \mid \\
&\quad \dots + c_k * \textbf{peek}(h_m, -1) > 0 \mid \\
&\quad B_1 \textbf{ or } B_2 \mid B_1 \textbf{ and } B_2 \\
E &::= C \mid \textbf{if } B \textbf{ then } E_1 \textbf{ else } E_2.
\end{aligned}$$

(target; P.I.D terms; PID controllers; boolean operators; real value; programs permitted by the sketch)

- Neurally Directed Program search (NDPS):

Algorithm 1 Neurally Directed Program Search

Input: POMDP M, neural policy $e_{\mathcal{N}}$, sketch $\mathcal{S}$
$\mathcal{H} \leftarrow \texttt{create\_histories}(e_{\mathcal{N}}, M)$
$e \leftarrow \texttt{initialize}(e_{\mathcal{N}}, \mathcal{H}, M, \mathcal{S})$
$R \leftarrow \texttt{collect\_reward}(e, M)$
repeat
  $(e', R') \leftarrow (e, R)$
  $\mathcal{H} \leftarrow \texttt{update\_histories}(e, e_{\mathcal{N}}, M, \mathcal{H})$ (a set of interesting input histories)
  $\mathcal{E} \leftarrow \texttt{neighborhood\_pool}(e)$ (generate a space of programs that are similar to e)
  $e \leftarrow \arg\min_{e' \in \mathcal{E}} \sum_{h \in \mathcal{H}} \|e'(h) - e_{\mathcal{N}}(h)\|$
  $R \leftarrow \texttt{collect\_reward}(e, M)$
until $R' \geq R$
Output: e'

the main intuition is that, the distance from $e_{\mathcal{N}}$ is a simpler objective than reward function.

Problem: the agent may encounter histories that are not possible under $e_{\mathcal{N}}$ or any of the programs encountered in previous iterations.

Solution: Input augmentation or periodic updates to the set $\mathcal{H}$.
→ every while we sample a set of additional histories by simulating the current programmatic policy, and add these samples to $\mathcal{H}$.

- The optimization: two steps;
  1. Enumerating a set of program templates (numerically parametrized programs) that are ① strictly similar to e and ② permitted by the sketch S, giving priority to shorter templates.
  2. Use Bayesian optimization to find the optimal parameters for the enumerated templates.
  - Bayesian optimization is better than SMT solving here.