Seminar: Few-Shot Bayesian Imitation Learning with Logical Program Policies

Yu-Zhe Shi

May 3, 2020
Author Profiles

Overview

Algorithm 1: LPP imitation learning

input: Demos $\mathcal{D}$, ensemble size $K$, max iters $L$
Create anti-demos $\overline{\mathcal{D}} = \{(s, a') : (s, a) \in \mathcal{D}, a' \neq a\}$;  
Set labels $y[(s, a)] = 1$ if $(s, a) \in \mathcal{D}$ else 0; 
Initialize approximate posterior $q$; 
for $i$ in 1, ..., $L$ do
  $f_i = \text{generate_next_feature}();$
  $X = \{(f_1(s, a), ..., f_i(s, a))^T : (s, a) \in \mathcal{D} \cup \overline{\mathcal{D}}\}$
  $\mu_i, w_i = \text{logical_inference}(X, y, p(f), K);$
  update_posterior($q, \mu_i, w_i$);
end
return $q$;
Dilemma of Imitation Learning

- **Behavior Cloning:** Overfitting, underconstrained policy class and weak prior.
- **Policy logical learning:** Need hand-crafted predicates, poor scalability.
- **Program synthesis:** Large search space.
Logical Program Policies

▶ "Top": Logical structure.
▶ "Bottom": Domain specific language expressions.
▶ Logically generate infinite policy classes from small scale DSL and score the candidates with likelihood and prior to prune searching space.
Prerequisites

- **Objective:** Given demo $\mathcal{D}$, learn policies $p(\pi|\mathcal{D})$
- $\mathcal{D} = (s_0, a_0, \ldots, s_{T-1}, a_{T-1}, s_T)$, states $s \in S$, actions $a \in A$
- Markov Process: $\mathcal{M} = (S, A, T, G)$ where $G \subset S$ is goal states, $T(s'|s, a)$ is transition distribution.
- State-conditional distribution over actions:
  \[ \pi(a|s) \in \Pi \]  
  where $\Pi$ is hypothesis classes.
- We learn $\pi^*$ which is optimal to $\mathcal{M}$. 
Policy classes

- Want to learn State-action classifiers:

\[ h : \mathcal{S} \times \mathcal{A} \rightarrow \{0, 1\} \] (2)

- \( h(s, a) = 0 \) action \( a \) never takes place when \( s \).
- \( h(s, a) = 1 \) action \( a \) may take place when \( s \).
- \( \pi(a|s) \propto h(s, a) \)
- \( \pi(a|s) \propto 1 \) when \( \forall a, h(s, a) = 0 \)
Bottom Level: Invent Predicates by Domain Specific Language

- Bottom Level: feature detection functions
  \( f \in \mathcal{H} : S \times A \rightarrow \{0, 1\} \).
- Input: \( s, a \).
- Output: Binary decision of whether \( a \) should take place when \( s \).
Top Level: Disjunctive Normal Form

\[ h(s, a) = \bigvee_{i=1}^{m} (\bigwedge_{j=1}^{n_i} f_{i,j}(s, a)) \]  

\[ h(s, a) = \bigvee_{i=1}^{m} \left( \bigwedge_{j=1}^{n_i} f_{i,j}(s, a)^{b_{i,j}} (1 - f_{i,j}(s, a))^{1-b_{i,j}} \right) \]  

where \( b_{i,j} \) determines whether \( f_{i,j} \) is negated.
<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell.is.value</td>
<td>V → C</td>
<td>Check whether the attended cell has a given value</td>
</tr>
<tr>
<td>shifted</td>
<td>O × C → C</td>
<td>Shift attention by an offset, then check a condition</td>
</tr>
<tr>
<td>scanning</td>
<td>O × C × C → C</td>
<td>Repeatedly shift attention by the given offset, and check which of two conditions is satisfied first</td>
</tr>
<tr>
<td>at.action.cell</td>
<td>C → P</td>
<td>Attend to the action cell and check a condition</td>
</tr>
<tr>
<td>at.cell.with.value</td>
<td>V × C → P</td>
<td>Attend to a cell with the value and check condition</td>
</tr>
</tbody>
</table>
Example of LPP

\[ h(s, a) = (f_{11}(s, a) \land f_{12}(s, a) \land \neg f_{13}(s, a)) \lor \\
(f_{11}(s, a) \land f_{22}(s, a) \land \neg f_{23}(s, a)) \]

- \( f_{11} = \text{at\_action\_cell(cell\_is\_value(\square))} \)
- \( f_{12} = \text{at\_action\_cell(shifted(\Leftrightarrow, cell\_is\_value(\square)))} \)
- \( f_{13} = \text{at\_action\_cell(shifted(\Rightarrow, cell\_is\_value(\square)))} \)
- \( f_{22} = \text{at\_action\_cell(shifted(\Leftrightarrow, cell\_is\_value(\square)))} \)
- \( f_{23} = \text{at\_action\_cell(shifted(\Rightarrow, cell\_is\_value(\square)))} \)
Imitation Learning

- Prior distribution of \( \pi \) over LPP:

\[
p(\pi) \propto \prod_{i=1}^{m} \prod_{j=1}^{n_i} p(f_{i,j}) \tag{5}
\]

where \( p(f) \) is a probabilistic context-free grammar, indicating how likely different rewritings are. The intuition is that we want to encode the prior with fewer and simpler \( f \)s.

- Likelihood \( p(\mathcal{D}|\pi) \) indicates the probabilistic of generating a demo \( \mathcal{D} \) from policies \( \pi \).

\[
p(\mathcal{D}|\pi) \propto \prod_{i=1}^{n} \prod_{j=1}^{T_i} \pi(a_{ij}|s_{ij}) \tag{6}
\]
### Production rule

<table>
<thead>
<tr>
<th>Programs</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \rightarrow \text{at_cell_with_value}(V, C)$</td>
<td>0.5</td>
</tr>
<tr>
<td>$P \rightarrow \text{at_action_cell}(C)$</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C \rightarrow \text{shifted}(O, B)$</td>
<td>0.5</td>
</tr>
<tr>
<td>$C \rightarrow B$</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Base conditions</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B \rightarrow \text{cell_is_value}(V)$</td>
<td>0.5</td>
</tr>
<tr>
<td>$B \rightarrow \text{scanning}(O, C, C)$</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O \rightarrow (N, 0)$</td>
<td>0.25</td>
</tr>
<tr>
<td>$O \rightarrow (0, N)$</td>
<td>0.25</td>
</tr>
<tr>
<td>$O \rightarrow (N, N)$</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Numbers</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N \rightarrow N$</td>
<td>0.5</td>
</tr>
<tr>
<td>$N \rightarrow -N$</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Natural numbers (for $i = 1, 2, \ldots$)</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N \rightarrow i$</td>
<td>$(0.99)(0.01)^i-1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Values (for each value $v$ in this game)</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V \rightarrow v$</td>
<td>$1/</td>
</tr>
</tbody>
</table>
Approximate the posterior

- $q$ is a weighted mixture of $K$ policies $\mu_1, \ldots, \mu_K$

\[
q(\pi) \approx p(\pi|D) \quad (7)
\]

- Minimize KL divergence $D_{KL}(q(\pi)|p(\pi|D))$

\[
q(\mu_j) = \frac{p(\mu_i|D)}{\sum_{i=1}^{K} p(\mu_i|D)} \quad (8)
\]
Training Algorithm

1. Given a set of demos $\mathcal{D}$ where $h(s, a) = 1$.
2. Generate negative samples

   $$\overline{\mathcal{D}} = \{(s, a')|(s, a) \in \mathcal{D}, a \neq a'\} \quad (9)$$

3. At iteration $i$, we use $i$ simplistic (i.e. of highest probability under $p(f)$) feature detectors $f_1, \ldots, f_i$ converting $(s, a)$ into

   $$\mathbf{x} \in \{0, 1\}^i = (f_1(s, a), \ldots, f_i(s, a))^T \quad (10)$$

4. A stochastic greedy decision-tree learner to learn a binary classifier $h(s, a)$.

5. Induce a candidate policy $\mu_* (a|s) \propto (s, a)$, calculate $p(\mu_*), p(\mathcal{D}|\mu_*)$ to decide whether to include $\mu_*$ into the mixture $q$. 
Inference

\[ \pi_*(s) = \arg_{a \in A} \max \mathbb{E}_{q}[\pi(a|s)] = \arg_{a \in A} \max \sum_{\mu \in q} q(\mu) \mu(a|s) \]  

(11)
Experiments
Experiments: Baseline Comparison

- Local Linear Network, Fully Connected Network, CNN: trained to classify whether each cell should be clicked based on 8 surrounding cells. Vanilla Program Induction: Policy Learning with brute force.
Ablation Study

<table>
<thead>
<tr>
<th>LPP</th>
<th>Nim</th>
<th>CT</th>
<th>Chase</th>
<th>STF</th>
<th>RFTS</th>
<th>Fence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features + NN</td>
<td>1.0</td>
<td>0.67</td>
<td>0.0</td>
<td>0.0</td>
<td>0.22</td>
<td>0.67</td>
</tr>
<tr>
<td>Features + NN + L₁ Reg</td>
<td>1.0</td>
<td>0.11</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>No Prior</td>
<td>1.0</td>
<td>0.44</td>
<td>0.78</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Sparsity Prior</td>
<td>1.0</td>
<td>0.78</td>
<td>1.0</td>
<td>0.78</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Summary and Inspiration

▶ Logical Program Policies: Reduce predicate invention into binary classification.
▶ Shared Domain Specific Language serves as meta-feature.
▶ Bayesian Imitation Learning: exploits probabilistic context-free grammar as priori to approximate posterior.