Differentiable Inductive Logic Programming for Structured Examples
(published: AAAI 2021)

Hikaru Shindo, Masaaki Nishino, Akihiro Yamamoto

Presented by:
Jesse Jing, Divyagna Bavikadi, Kaustuv Mukherji
For a student led discussion on:
Neuro-symbolic Reasoning
At Arizona State University
Oct 24, 2022
Logic Programming Basics

$\text{Language } L = (\mathcal{P}, \mathcal{F}, \mathcal{A}, \mathcal{V})$

$\mathcal{P} : \text{Set of Predicates } (p/n)$

$\mathcal{F} : \text{Set of Function Symbols } (f/n)$

$\mathcal{A} : \text{Set of Constants}$

$\mathcal{V} : \text{Set of Variables}$
Logic Programming Basics

A term \((t)\) could be a constant, a variable, or \(f(t_1, t_2, \ldots, t_n)\).

An atom is a formula \(p(t_1, t_2, \ldots, t_n)\).

A ground atom has no variables.

A clause is a disjunction (\(\lor\)) of literals.

\(V(C)\) : Set of variables in clause \(C\).
Logic Programming Basics

\( \theta \): Substitution operation \([x_i \leftarrow t_i]\)

\( \theta \) applied to head of a rule \( A \), is written as \( A\theta \).

A unifier for a set \( \{A_1, A_2, \ldots, A_n\} \) is a substitution \( \theta \) such that, \( A_1\theta = A_2\theta = \cdots = A_n\theta \), written as: \( \theta = \sigma(\{A_1, A_2, \ldots, A_n\}) \)

\( \sigma \): Unification function

\( \bar{\sigma}(\{A_1, A_2, \ldots, A_n\}) = T \) if set is unifiable, else \( \perp \)

Decision function
The *ILP* problem

Given:

\[ Q = (\mathcal{E}^+, \mathcal{E}^-, \mathcal{B}, \mathcal{L}) \]

Find a set of definite clauses (one atom in the head) \( \mathcal{H} \subseteq \mathcal{L} \), such that,

\[ \forall A \in \mathcal{E}^+ \mathcal{H} \cup \mathcal{B} \models A. \]
\[ \forall A \in \mathcal{E}^- \mathcal{H} \cup \mathcal{B} \not\models A. \]
Why $\partial ILP$?

- Unlike ILP, robust to noise and training data\(^1\)

- Formulates ILP problems as an optimization problem, which can be differentiated and solved much like traditional machine learning methods. Hence, it’s easy to combine these approaches to neural systems.

---

Drawbacks of previous $\partial ILP$ systems

- No function symbols are allowed.
- The arity of predicates must be less than 2.
- The number of atoms in the clause body must not exceed 2.
- Every program must be comprised of pairs of rules for each predicate.
Drawbacks of previous $\delta ILP$ systems

Thus,

- Unsuitable for complex structured data, such as sequences or trees.

- Unsuitable for complex programs that are comprised of several clauses for a predicate.
Today’s paper -

Differentiable Inductive Logic Programming for Structured Examples

Hikaru Shindo, Masaaki Nishino, Akihiro Yamamoto
What are Structured Examples?

• Examples of data which has a ‘structure’ to it. 
  \[ [a, b, c], [d], [a, b, c, d] \]

• Sequences and Trees  
  \[ [1, 2, 4, 8, ...] \]

• Data with function symbols  
  \[ f^4(x) \ast f^8(x) = f^{12}(x) \]
Challenges

• Consider a large number of clauses.

• Number of possible ground atoms are infinite.
  \[ c_1, c_2, \ldots, c_n, f(c_1), g(c_1), h(c_1, c_2) \ldots \ldots \ldots \]

• Keep memory and computation costs manageable.
What should the result look like?

\[ \mathcal{E}^+ = \{p(a, a), p(b, b), p(b, c), p(c, b)\} \]

\[ \mathcal{E}^- = \{p(a, b)\} \]

\[ \mathcal{B} = \{q(b, c), q(c, b)\} \]
What should the result look like?

\[ \mathcal{E}^+ = \{ p(a, a), p(b, b), p(b, c), p(c, b) \} \]
\[ \mathcal{E}^- = \{ p(a, b) \} \]

\[ \mathcal{B} = \{ q(b, c), q(c, b) \} \]

One possible solution:
\[ p(x, x) \]
What should the result look like?

\[ \mathcal{E}^+ = \{p(a, a), p(b, b), p(b, c), p(c, b)\} \]
\[ \mathcal{E}^- = \{p(a, b)\} \]

\[ \mathcal{B} = \{q(b, c), q(c, b)\} \]

One possible solution:
\[ p(x, x) \]
\[ p(x, y) \leftarrow q(x, y) \]
The Building Blocks

1. Get Clauses

2. Determine Ground Atoms for clauses

3. Introduce weights for clauses

4. Do Inference: Minimize error (Differentiable!)
Clause Search with Refinement

- **Objective:** Find promising clauses that entail many positive examples but few negative examples.

  - **Start:** General/Strong clauses
    Might entail many examples, including negative examples.

  - **Refine:** Iteratively specify/weaken clauses.
    Now, they may entail lesser number of $\mathcal{E}^+$, but will also entail lesser $\mathcal{E}^-$. 
The Refinement Operator ($\rho_\mathcal{L}$)

$$\rho_\mathcal{L}(C) = \rho^\text{func}_\mathcal{L}(C') \cup \rho^\text{subs}_\mathcal{L}(C') \cup \rho^\text{rep}_\mathcal{L}(C) \cup \rho^\text{add}_\mathcal{L}(C)$$
The Refinement Operator ($\rho_L$)

$$\rho_L(C) = \rho_L^{\text{func}}(C') \cup \rho_L^{\text{subs}}(C') \cup \rho_L^{\text{rep}}(C) \cup \rho_L^{\text{add}}(C)$$

Start: $p(x, y)$

$\mathcal{V} = \{x, y, z\}$

$\mathcal{A} = \{a, b\}$

- Application of function symbols

$$\rho_L^{\{\text{func}\}}(p(x, y)) = p(x, f(z))$$
The Refinement Operator ($\rho_L$)

$$\rho_L(C) = \rho^\text{func}_L(C') \cup \rho^\text{subs}_L(C) \cup \rho^\text{rep}_L(C) \cup \rho^\text{add}_L(C)$$

Start: $p(x, y)$

$\mathcal{V} = \{x, y, z\}$

$\mathcal{A} = \{a, b\}$

• Application of function symbols

$$\rho^\{func\}_L(p(x, y)) = p(x, f(z))$$

• Substitution of constants

$$\rho^\{subs\}_L(p(x, y)) = p(x, a)$$
The Refinement Operator \((\rho_{\mathcal{L}})\)

\[
\rho_{\mathcal{L}}(C) = \rho_{\mathcal{L}}^{\text{func}}(C) \cup \rho_{\mathcal{L}}^{\text{subs}}(C) \cup \rho_{\mathcal{L}}^{\text{rep}}(C) \cup \rho_{\mathcal{L}}^{\text{add}}(C)
\]

Start: \(p(x, y)\)

\[\mathcal{V} = \{x, y, z\}\]
\[\mathcal{A} = \{a, b\}\]

- Replacement of variables

\[
\rho_{\mathcal{L}}^{\{\text{rep}\}}(p(x, y)) = p(x, z)
\]
The Refinement Operator ($\rho_\mathcal{L}$)

$$\rho_\mathcal{L}(C) = \rho_\mathcal{L}^{\text{func}}(C) \cup \rho_\mathcal{L}^{\text{subs}}(C) \cup \rho_\mathcal{L}^{\text{rep}}(C) \cup \rho_\mathcal{L}^{\text{add}}(C)$$

Start: $p(x, y)$

- Replacement of variables
  $$\rho_\mathcal{L}^{\{\text{rep}\}}(p(x, y)) = p(x, z)$$

- Addition of atoms
  $$\rho_\mathcal{L}^{\{\text{add}\}}(p(x, y)) = p(x, y) \leftarrow q(x, y)$$

\[ \mathcal{V} = \{x, y, z\} \]
\[ \mathcal{A} = \{a, b\} \]
What should the result look like?

\[ \mathcal{E}^+ = \{p(a, a), p(b, b), p(b, c), p(c, b)\} \]
\[ \mathcal{E}^- = \{p(a, b)\} \]
\[ \mathcal{B} = \{q(b, c), q(c, b)\} \]
Refinement with beam search

\[ \mathcal{E}^+ = \{p(a, a), p(b, b), p(b, c), p(c, b)\} \]
\[ \mathcal{E}^- = \{p(a, b)\} \]

\[ \mathcal{B} = \{q(b, c), q(c, b)\} \]

\[ C_0 = \{p(x, y)\} \]

\[ N_{beam} = 2 \]
\[ T_{beam} = 2 \]
Refinement with beam search

\[
eval(R, Q) = \left| \{ E \mid E \in \mathcal{E}^+ \land B \cup \{R\} \models E \} \right|
\]
Adaptive Fact Enumeration
Soft Program Composition
Putting it all together – solving the ILP!
Experiments

• Enumeration Algorithm
• Beam Searching Approach for Clause Generation
• Memory and Computational efficiency
• Does the method efficiently learn from noisy and structured examples?
• Used standard 5 ILP tasks with structured examples
Experiments

\[ V = \{x, y, z, v, w\} \quad \gamma = 10^{-5} \]

- Member:

\[ \mathcal{P} = \{\text{mem}/2\} \quad \mathcal{F} = \{f/2\} \]

\[ \mathcal{C}_0 = \{\text{mem}(x, y)\} \quad \mathcal{A} = \{a, b, c, *\} \]

\[ (N_{\text{beam}}, T_{\text{beam}}) = (3, 3) \quad m = 2 \quad T = 4 \]

\[ \mathcal{E}^+ = \{\text{mem}(a, [a, c]), \text{mem}(a, [b, a]), \ldots\}, \]

\[ \mathcal{E}^- = \{\text{mem}(c, [b, a]), \text{mem}(c, [a]), \ldots\}, \]

\[ \mathcal{B} = \{\text{mem}(a, [a]), \text{mem}(b, [b]), \text{mem}(c, [c])\}. \]
Experiments

• Plus:
  \[ \mathcal{P} = \{\text{plus}/3\}, \mathcal{F} = \{s/1\} \]

  \[ \mathcal{C}_0 = \{\text{plus}(x, y, z)\} \quad \mathcal{A} = \{0\} \]

  \[ (N_{\text{beam}}, T_{\text{beam}}) = (10, 5) \quad m = 3 \quad T = 8 \]

  \[ \mathcal{E}^+ = \{\text{plus}(s(0), 0, s(0)), \text{plus}(s^5(0), s^3(0), s^8(0)), \ldots\}, \]

  \[ \mathcal{E}^- = \{\text{plus}(s(0), s^2(0), 0), \text{plus}(0, s^2(0), s^4(0)), \ldots\}, \]

  \[ \mathcal{B} = \{\text{plus}(0, 0, 0)\}. \]
Experiments

- Append:

\[ P = \{app/3\}, \ F = \{f/2\} \]

\[ A = \{a, b, c, *\} \quad C_0 = \{app(x, y, z)\} \]

\[ (N_{beam}, T_{beam}) = (10, 5) \quad m = 3 \quad T = 4 \]

\[ \mathcal{E}^+ = \{app([c], [], [c]), app([a, a, b], [b, c], [a, a, b, b, c]), \ldots\} \]

\[ \mathcal{E}^- = \{app([], [a, a], [a, a, b]), app([b], [], [c]), \ldots\} \]

\[ \mathcal{B} = \{app([], [], [])\} \]
Experiments

• Delete:

\[ P = \{ \text{del}/3 \}, \quad F = \{ f/2 \} \]

\[ C_0 = \{ \text{del}(x, y, z) \} \quad A = \{ a, b, c, * \} \]

\[ (N_{beam}, T_{beam}) = (10, 5) \quad m = 2 \quad T = 4 \]

\[ E^+ = \{ \text{del}(b, [a, c, b], [a, c]), \text{del}(a, [b, a, a], [b, a]), \ldots \}, \]

\[ E^- = \{ \text{del}(c, [c, a, a], [a, b]), \text{del}(b, [b], [a]), \ldots \}, \]

\[ B = \{ \text{del}(a, [a], []), \text{del}(b, [b], []), \text{del}(c, [c], []) \}. \]
Experiments

• Subtree:

\[ \mathcal{P} = \{ \text{sub}/2 \} \quad \mathcal{F} = \{ f/2 \} \]

\[ \mathcal{C}_0 = \{ \text{sub}(x, y) \} \quad \mathcal{A} = \{ a, b, c \} \]

\[ (N_{\text{beam}}, T_{\text{beam}}) = (15, 3) \quad m = 4 \quad T = 4 \]

\[ \mathcal{E}^+ = \{ \text{sub}(f(b, b), f(f(f(b, b), f(a, c)), f(a, c))), \ldots \}, \]

\[ \mathcal{E}^- = \{ \text{sub}(f(a, a), f(f(c, a), f(a, c))), \ldots \}, \]

\[ \mathcal{B} = \{ \text{sub}(a, a), \text{sub}(b, b), \text{sub}(c, c) \}. \]
Experiment 1

- The dILP method won’t be feasible here.
- Enumeration algorithm yields reasonable number of ground atoms

<table>
<thead>
<tr>
<th>Member</th>
<th>Plus</th>
<th>Append</th>
<th>Delete</th>
<th>Subtree</th>
</tr>
</thead>
<tbody>
<tr>
<td>228</td>
<td>1857</td>
<td>2899</td>
<td>2513</td>
<td>2172</td>
</tr>
</tbody>
</table>

Table 1: Number of enumerated ground atoms
Experiment 2

- Two Clause Generation Algorithms:
- 1. Beam Searching + Refinement
- 2. Naive generation without beam searching

AUC for number of generated clauses
Experiment 3

• 1. Multiple weights and softer approach $\mathbf{W} \in \mathbb{R}^{|C|}$
• 2. 2-d weights for pairs of clauses
• assigned weights in the form of a 2-d matrix $\mathbf{W} \in \mathbb{R}^{|C| \times |C|}$

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Proposed</th>
<th>Pair</th>
<th>Runtime [s] Proposed</th>
<th>Pair</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member</td>
<td>24</td>
<td>144</td>
<td>0.015</td>
<td>0.12</td>
</tr>
<tr>
<td>Plus</td>
<td>120</td>
<td>1600</td>
<td>0.03</td>
<td>6.91</td>
</tr>
<tr>
<td>Append</td>
<td>150</td>
<td>2500</td>
<td>0.09</td>
<td>5.18</td>
</tr>
<tr>
<td>Delete</td>
<td>150</td>
<td>2500</td>
<td>0.06</td>
<td>5.4</td>
</tr>
<tr>
<td>Subtree</td>
<td>80</td>
<td>400</td>
<td>0.039</td>
<td>1.11</td>
</tr>
</tbody>
</table>

Table 2: Number of parameters and mean runtime in learning steps
**Experiment 4**

- Change the proportion of the mislabeled training data.
- Mean-squared test error as proportion of mislabeled training data: (gets a solution for mislabeled data as well)
<table>
<thead>
<tr>
<th>Problem</th>
<th>Learned logic program</th>
</tr>
</thead>
</table>
| Member    | `mem(x, [y | z]) ← mem(x, z)`  
              `mem(x, [x | y])`  |
| Plus      | `plus(0, x, x)`  
              `plus(x, s(y), s(z)) ← plus(x, y, z)`  
              `plus(s(x), y, s(z)) ← plus(y, x, z)` |
| Append    | `app([], x, x)`  
              `app(x, [], x)`  
              `app([x | y], z, [x | v]) ← app(y, z, v)` |
| Delete    | `del(x, [x | y], y)`  
              `del(x, [y | z], [y | v]) ← del(x, z, v)` |
| Subtree   | `sub(f(x, y), f(x, y))`  
              `sub(x, f(y, z)) ← sub(x, z)`  
              `sub(x, f(y, z)) ← sub(x, y)`  
              `sub(x, f(y, x))` |

Table 3: Learned logic programs in standard ILP tasks
Conclusion

• Differentiable inductive logic programming framework that deals with complex logic programs that have several clauses with function symbols that yield readable outputs for noisy structured data.

• Clause generation algorithm uses beam searching with refinement that improves differential program searching

• Enumeration algorithm for sufficient ground atoms from given examples and clauses

• Soft program composition approach using multiple weights and the softer function

• Proposed method estimates logic programs efficiently in terms of memory and computational costs
Conclusion

• Filling the gaps for both $\partial$ILP and standard ILP approaches, the framework learns logic programs successfully from noisy and structured examples

• Incorporates symbolic methods, such as refinement, with a differentiable ILP approach
Limitations

• Scalability for large-scale programs.
• Learning more expressive programs for more complex tasks like learning sorting from captured images. (Incorporating biases\textsuperscript{2} to manage the search space can help solve this)

\textsuperscript{2}Claire, N.; Celine, R.; Hilde, A.; Francesco, B.; and Birgit, ´ T. 1996. Declarative bias in ILP. Advances in inductive logic programming 32: 82—103
Algorithm 1 Clause generation by beam searching

Input: $C_0$, $Q$, $N_{beam}$, $T_{beam}$

1: $C_{to\_open} \leftarrow C_0$
2: $C \leftarrow \emptyset$
3: $t = 0$
4: while $t < T_{beam}$ do
5:     $C_{beam} \leftarrow \emptyset$
6:     for $C_i \in C_{to\_open}$ do
7:          $C = C \cup \{C_i\}$
8:     for $R \in \rho_C(C_i)$ do
9:         score = eval($R$, $Q$) //Evaluate each clause
10:        $C_{beam} = \text{insert}(C_{beam}, R, score)$ //Insert refined clause in order of scores possibly discarding it
11:     $C_{to\_open} = C_{beam}$ //top-$N_{beam}$ clauses are refined in the next loop
12:     $t = t + 1$
13: return $C$
Algorithm 2 Enumeration of ground atoms

Input: \( Q, C, T \)

1: \( G \leftarrow \{ \bot, \top \} \cup \varepsilon^+ \cup \varepsilon^- \cup B \)
2: \textbf{for} \( i = 0 \) to \( T - 1 \) \textbf{do}
3: \hspace{1em} \( S \leftarrow \emptyset \)
4: \hspace{2em} \textbf{for} \( A \leftarrow B_1, \ldots, B_n \) in \( C \) \textbf{do}
5: \hspace{3em} \textbf{for} \( G \in G \) \textbf{do}
6: \hspace{4em} \textbf{if} \( \bar{\sigma}(A, G) \) \textbf{then}
7: \hspace{5em} \theta \leftarrow \sigma(A, G)
8: \hspace{6em} S \leftarrow S \cup \{ B_1\theta, \ldots, B_n\theta \}
9: \hspace{2em} G \leftarrow G \cup S
10: \textbf{return} \( G \)