Differentiable Inductive Logic Programming
Overview

- ILP Background
- ILP by SAT-solving
- Neural Architecture
- Complexity
- Experiments
ILP

Background
Logic

First-Order Logic (predicates and constants)

- Given predicates $p, q$ constant $c$ and variable $x$
- $p(c)$ is a ground atom
- $p(x)$ is non-ground
- $q(x) \leftarrow p(x)$ is a non-ground rule (“if $p$ then $q$”)
  - Each non-ground rule will have multiple grounded instances
- Assume facts (ground atoms) and (non-ground) rules
- Limits on the number of inference steps (i.e., applications of a fixpoint operator) – denoted $T$
Predicates

• Predicates have up to two arguments
  • i.e., arity of 2
  • This is a common assumption we have seen in other work

• Predicates are either extensional (never appear in rule head) or intentional (may appear in the rule head)
  • Some intentional predicates may be “invented predicates” created during the learning process (the amount of these can be treated as a hyperparameter)

• The “target predicate” is the predicate that we wish to predict
ILP Problem

- Inductive Logic Programming (ILP) problem is formally defined

- Tuple \((\mathcal{B}, \mathcal{P}, \mathcal{N})\) where:
  - \(\mathcal{B}\) is a set of facts (ground atoms)
  - \(\mathcal{P}\) is a set of positive ground atoms (formed with the target predicate)
  - \(\mathcal{N}\) is the set of negative ground atoms (formed with the target predicate)

- The authors point out that \((\mathcal{B}, \mathcal{P}, \mathcal{N})\) essentially specifies a world

- A solution to an ILP problem is a set of rules \(R\) such that
  - \(\forall a \in \mathcal{P}, \mathcal{B} \cup \mathcal{R} \models a\)
  - \(\forall a \in \mathcal{N}, \mathcal{B} \cup \mathcal{R} \not\models a\)

- In the paper, the authors use multiple ILP problem to conduct ILP, sample from each, and optimize a loss function
ILP as Satisfiability
Approach: ILP as Satisfiability

• A template is a way to generate “clauses”
  • Here, a clause can be thought of as a candidate rule, and template is a way to create that rule

• Given an ILP problem, use a template to generate a set of clauses

• Add a Boolean flag to each template

• Find a set assignments to the Boolean flags such that the resulting clauses (with positive flags), unioned with the facts entail the positive examples and do not entail the negative ones

• This concept was previously introduced, but not solved using neural architecture
Example

• To illustrate this concept, we work through a simple example.
• Here we have a target predicate, an extensional predicate, and four constants

\[
\text{target} = q/2
\]
\[
P_e = \{ p/2 \}
\]
\[
C = \{ a, b, c, d \}
\]
Example:
ILP Problem Instance

- **Ground Facts**
  \[ B = \{ p(a, b), p(b, c), p(c, d) \} \]

- **Positive Examples**
  \[ P = \{ q(a, b), q(a, c), q(a, d), q(b, c), q(b, d), q(c, d) \} \]

- **Negative Examples**
  \[ N = \{ q(X, Y) \mid (X, Y) \in \{a, b, c, d\}^2, q(X, Y) \notin P \} \]

Taken from Appendix B of Evans & Grefenstette, 2018
Example:
Template #1 and Generated Clauses

- Template 1 allows no existentially quantified variables in the body and no intentional predicates in the body
- All 8 clauses generated by the template are shown
- Clause generation causes some redundancies which can be easily resolved at time of generation

\[ \tau_q^1 \text{ is } (v = 0, int = 0) \]

1. \( q(X, Y) \leftarrow p(X, X), p(X, Y) \)
2. \( q(X, Y) \leftarrow p(X, X), p(Y, X) \)
3. \( q(X, Y) \leftarrow p(X, X), p(Y, Y) \)
4. \( q(X, Y) \leftarrow p(X, Y), \underline{p(X, Y)} \)
5. \( q(X, Y) \leftarrow p(X, Y), p(Y, X) \)
6. \( q(X, Y) \leftarrow p(X, Y), p(Y, Y) \)
7. \( q(X, Y) \leftarrow p(Y, X), \underline{p(Y, X)} \)
8. \( q(X, Y) \leftarrow p(Y, X), p(Y, Y) \)

Taken from Appendix B of Evans & Grefenstette, 2018
Example:
Template #2 and Generated Clauses

- Template 2 allows one existentially quantified variable in the body and one intentional predicate in the body
- 58 clauses are generated (only 16 are shown)

\[ \tau_q^2 \text{ is } (v = 1, \text{int} = 1) \]

1. \( q(X,Y) \leftarrow p(X,X), q(Y,X) \)
2. \( q(X,Y) \leftarrow p(X,X), q(Y,Y) \)
3. \( q(X,Y) \leftarrow p(X,X), q(Y,Z) \)
4. \( q(X,Y) \leftarrow p(X,X), q(Z,Y) \)
5. \( q(X,Y) \leftarrow p(X,Y), q(X,X) \)
6. \( q(X,Y) \leftarrow p(X,Y), q(X,Z) \)
7. \( q(X,Y) \leftarrow p(X,Y), q(Y,X) \)
8. \( q(X,Y) \leftarrow p(X,Y), q(Y,Y) \)
9. \( q(X,Y) \leftarrow p(X,Y), q(Y,Z) \)
10. \( q(X,Y) \leftarrow p(X,Y), q(Z,X) \)
11. \( q(X,Y) \leftarrow p(X,Y), q(Z,Y) \)
12. \( q(X,Y) \leftarrow p(X,Y), q(Z,Z) \)
13. \( q(X,Y) \leftarrow p(X,Z), q(Y,X) \)
14. \( q(X,Y) \leftarrow p(X,Z), q(Y,Y) \)
15. \( q(X,Y) \leftarrow p(X,Z), q(Y,Z) \)
16. \( q(X,Y) \leftarrow p(X,Z), q(Z,Y) \)

Taken from Appendix B of Evans & Grefenstette, 2018
Example:
Boolean Flags Associated with Generated Clauses

• We specify the set of Boolean flags

\[ \Phi = \left\{ f_{p,i,j} \right\}_{p \in P_i, i \in \{1,2\}, j = 1..|cl(\tau_p^i)|} \]

• A clause is part of the solution iff the associated flag is true

\[ f_{p,i,j} \text{ iff } C_{p,i,j} \in R \]

• The authors look for two clauses per target predicate – this number affects the complexity

Taken from Appendix B of Evans & Grefenstette, 2018
Example:
Template #1 and Generated Clauses with Boolean Flags

\[ \tau_q^1 \text{ is } (v = 0, int = 0) \]

1. \( q(X, Y) \leftarrow p(X, X), p(X, Y), f_q^{1,1} \)
2. \( q(X, Y) \leftarrow p(X, X), p(Y, X), f_q^{1,2} \)
3. \( q(X, Y) \leftarrow p(X, X), p(Y, Y), f_q^{1,3} \)
4. \( q(X, Y) \leftarrow p(X, Y), p(X, Y), f_q^{1,4} \)
5. \( q(X, Y) \leftarrow p(X, Y), p(Y, X), f_q^{1,5} \)
6. \( q(X, Y) \leftarrow p(X, Y), p(Y, Y), f_q^{1,6} \)
7. \( q(X, Y) \leftarrow p(Y, X), p(Y, X), f_q^{1,7} \)
8. \( q(X, Y) \leftarrow p(Y, X), p(Y, Y), f_q^{1,8} \)

Taken from Appendix B of Evans & Grefenstette, 2018
Example:
Template #2 and Generated Clauses with Boolean Flags

Only 16 of the 58 clauses are shown

\[ \tau_q^2 \text{ is } (v = 1, \text{int} = 1) \]

1. \( q(X, Y) \leftarrow p(X, X), q(Y, X), f_q^{2,1} \)
2. \( q(X, Y) \leftarrow p(X, X), q(Y, Y), f_q^{2,2} \)
3. \( q(X, Y) \leftarrow p(X, X), q(Y, Z), f_q^{2,3} \)
4. \( q(X, Y) \leftarrow p(X, X), q(Z, Y), f_q^{2,4} \)
5. \( q(X, Y) \leftarrow p(X, Y), q(X, X), f_q^{2,5} \)
6. \( q(X, Y) \leftarrow p(X, Y), q(X, Z), f_q^{2,6} \)
7. \( q(X, Y) \leftarrow p(X, Y), q(Y, X), f_q^{2,7} \)
8. \( q(X, Y) \leftarrow p(X, Y), q(Y, Y), f_q^{2,8} \)
9. \( q(X, Y) \leftarrow p(X, Y), q(Y, Z), f_q^{2,9} \)
10. \( q(X, Y) \leftarrow p(X, Y), q(Z, X), f_q^{2,10} \)
11. \( q(X, Y) \leftarrow p(X, Y), q(Z, Y), f_q^{2,11} \)
12. \( q(X, Y) \leftarrow p(X, Y), q(Z, Z), f_q^{2,12} \)
13. \( q(X, Y) \leftarrow p(X, Z), q(Y, X), f_q^{2,13} \)
14. \( q(X, Y) \leftarrow p(X, Z), q(Y, Y), f_q^{2,14} \)
15. \( q(X, Y) \leftarrow p(X, Z), q(Y, Z), f_q^{2,15} \)
16. \( q(X, Y) \leftarrow p(X, Z), q(Z, Y), f_q^{2,16} \)

Taken from Appendix B of Evans & Grefenstette, 2018
Selecting Boolean Variables

• We can now formulate the ILP problem as a satisfiability problem – where we identify two Booleans per intentional predicate such that the positive examples of the target are entailed and the negative examples are not.

\[ \forall p \in P, \forall i \in \{1, 2\}, \exists j \in \{1..|cl(\tau^i_p)|\} \; f^{i,j}_p \in F \]
\[ B \cup F, \bigcup_{p \in P, i \in \{1,2\}} \text{cl}^*(\tau^i_p) \models \gamma \text{ for all } \gamma \in \mathcal{P} \]
\[ B \cup F, \bigcup_{p \in P, i \in \{1,2\}} \text{cl}^*(\tau^i_p) \not\models \gamma \text{ for all } \gamma \in \mathcal{N} \]

• Based on the selection of Boolean variables, we can create the set of rules by simply selecting the associated clauses

\[ R = \{ C^{i,j}_p \mid f^{i,j}_p \in F \} \]
Example: Identifying the solution

• In our example, only one pair of Boolean variables solves the satisfiability problem

\[ F = \{ f_q^{1,4}, f_q^{2,16} \} \]

• This pair is associated with the following clauses

\[ R = \{ C_q^{1,4}, C_q^{2,16} \} \]

• Which gives us the following non-ground rules in the resulting logic program

\[
q(X, Y) \leftarrow p(X, Y), p(X, Y) \\
q(X, Y) \leftarrow p(X, Z), q(Z, Y)
\]

Taken from Appendix B of Evans & Grefenstette, 2018
Neural Architecture
Learning Problem

• The overall goal is to compute the following conditional probability

$$ p(\lambda | \alpha, W, \Pi, \mathcal{L}, \mathcal{B}) $$

Truth value for ground atom $\alpha$

Weights (learned), program template, language, and facts

• Such that the following loss is minimized

$$ \text{loss} = - \mathbb{E}_{(\alpha, \lambda) \sim \Lambda} [\lambda \cdot \log p(\lambda | \alpha, W, \Pi, \mathcal{L}, \mathcal{B}) + (1 - \lambda) \cdot \log (1 - p(\lambda | \alpha, W, \Pi, \mathcal{L}, \mathcal{B}))] $$

Atom, value (in $\{0,1\}$)
pairs in ground truth

Notation from Evans & Grefenstette, 2018
High-Level Constructs

• The conditional probability is computed based on four functions.

\[ p(\lambda \mid \alpha, W, \Pi, \mathcal{L}, \mathcal{B}) = f_{\text{extract}}(f_{\text{infer}}(f_{\text{convert}}(\mathcal{B}), f_{\text{generate}}(\Pi, \mathcal{L}), W, T), \alpha) \]

• Two of them are straight-forward translations between vector and symbol representations

\[ f_{\text{extract}}(x, \gamma) = x[\text{index}(\gamma)] \quad f_{\text{convert}}(\mathcal{B}) = y \quad \text{where} \quad y[i] = \begin{cases} 1 & \text{if } \gamma_i \in \mathcal{B} \\ 0 & \text{otherwise} \end{cases} \]

• The third generates the clauses from the templates

\[ f_{\text{generate}}(\Pi, \mathcal{L}) = \{ cl(\tau^i_p) \mid p \in P_i, i \in \{1, 2\} \} \]

Notation from Evans & Grefenstette, 2018
Overall Architecture

Figure from Evans & Grefenstette, 2018
Inference in the Learning Architecture

• Given a pair of rule clauses, they update the truth value for an atom based on the maximum. Note they consider all pairs

\[ G_p^{j,k}(a) = x \quad \text{where} \quad x[i] = \max \left( F_p^{1,j}(a)[i], F_p^{2,k}(a)[i] \right) \]

• The below vector stores the values for each ground atom. It is initialized by the facts. The subscript indicates the number of inference steps (similar to applications of an fixpoint operator)

\[ a_0[x] = \begin{cases} 1 & \text{if } \gamma_x \in B \\ 0 & \text{otherwise} \end{cases} \]

Notation from Evans & Grefenstette, 2018
Inference in the Learning Architecture

• The below vector is the output of one inference step when clause pair $j, k$ is applied for returning predicate $p$

$$c^p_{t,j,k} = g^j_k(a_t)$$

• The value for predicate $p$ is based on a probability-weighted combination of the above vectors

$$b^p_t = \sum_{j,k} c^p_{t,j,k} \cdot \frac{e^{w_p[j,k]}}{\sum_{j',k'} e^{w_p[j',k']}}$$

Notation from Evans & Grefenstette, 2018
Inference in the Learning Architecture

• Note that vector $b^p$ only has non-zero entries for predicate $p$. So, we use it to update vector $a$ at each inference step

$$a_{t+1} = f_{\text{amalgamate}}(a_t, \sum_{p \in P_i} b^p_t)$$

• The authors tested max for amalgamation but found probabilistic sum to work better with gradient descent

$$f_{\text{amalgamate}}(x, y) = \max(x, y) \quad f_{\text{amalgamate}}(x, y) = x + y - x \cdot y$$

Notation from Evans & Grefenstette, 2018
Selection of Fuzzy Operator

• The authors investigated several methods for combining atoms with fuzzy conjunction
• Similar to results on LTN, they found product t-norm provided the best performance

Godel t-norm: $x \ast y = min(x, y)$

Łukasiewicz t-norm: $x \ast y = max(0, x + y - 1)$

Product t-norm: $x \ast y = x \cdot y$

Notation from Evans & Grefenstette, 2018
Complexity Analysis
Complexity

• Primary and secondary collection vectors require the following amount of weights

\[ 2 \cdot n \cdot t \cdot \sum_{i=1}^{\text{num}_r} \prod_{j} |\text{cl}(\tau_i^j)| \]

\[ 3 \cdot n \cdot |C| \cdot \sum_{i=1}^{\text{num}_r} \prod_{j} |\text{cl}(\tau_i^j)| \]

• Here \( n \) is the number of ground atoms expressed as follows

\[ n = |P| \cdot |C|^2 + 1 \]
Bounding the Number of Templates

- A major source of complexity is the number of templates, which can be bounded by various quantities as follows:

\[
\sum_{i=1}^{\lvert P_{\text{in}} \rvert} \prod_j \lvert cl(\tau_i^j) \rvert \leq \sum_{i=1}^{\lvert P_{\text{in}} \rvert \cdot \text{num}_\tau} \prod_j (\lvert P_{\text{ext}} \rvert + \text{int}_i) (\text{arity}_i + v)_{\text{arity}_{\text{max}}}^{\text{body}_{\text{max}}} \\
\leq \lvert P_{\text{in}} \rvert \cdot (\lvert P \rvert (\text{arity}_{\text{max}} + v)_{\text{arity}_{\text{max}}}^{\text{body}_{\text{max}}} \cdot \text{num}_\tau)
\]

- Under the assumptions in the paper, the above bound is equal to the following:

\[6561 \cdot \lvert P_{\text{in}} \rvert \cdot |P|^4\]
Overall Complexity

• Considering the number of templates, the overall number of floats can be bounded as follows:

\[ 6561 \cdot (2 \cdot n \cdot t + 3 \cdot n \cdot |C|) \cdot |P_{in}| \cdot |P|^4 \]
\[ \leq 6561 \cdot (|P| \cdot |C|^2 + 1) \cdot (2 \cdot t + 3 \cdot |C|) \cdot |P_{in}| \cdot |P|^4 \]
\[ \approx 6561 \cdot (2 \cdot t + 3 \cdot |C|) \cdot |P_{in}| \cdot |C|^2 \cdot |P|^5 \]
\[ \leq K \cdot |P_{in}| \cdot |C|^3 \cdot |P|^5 \]
Experimental Evaluation
Experimental Notes

• Training data:
  • Training data consists of multiple \((B, P, N)\) triples
  • At each step, one of the triples is sampled
  • From the sampled triple, a mini batch of \(P \cup N\) is selected
  • Authors state that this method helps escape local minima
  • Training occurs in 6,000 steps

• Other notes:
  • Cross-entropy loss (seen in other work as well)
  • RMS Prop used as optimizer with a learning rate of 0.5
    • Adam also gave reasonable results
    • RMS Prop performed well with lower learning rates (e.g., 0.01)
  • Clause weights initialized from a normal distribution over the interval \([0,1]\) (mean zero, sd between 0 and 2)
Multiple Runs / Local Minima Issues

- Now $\delta$ilp does not always find the correct solution. All existing neural program induction systems are susceptible to getting stuck in local minima when they are started with an unfortunate random initialisation of initial weights.

- (This is a result of the “spikiness” of the loss landscape in program induction tasks).

- $\delta$ilp is no exception, and on complex problems it only finds a correct solution a certain proportion of the time.

- However, this dependence on initial random weights is not a serious problem in this particular system, because we can model-select based on the training data. After a certain number (e.g. 100) of runs, we look at the loss on the training data, and choose the model with the lowest loss.

- Even though we are model-selecting on training data, the top-scoring model will generalise robustly to unseen test data because generalisation has been baked into the system from the beginning. The only possible rules it can learn are universally quantified rules.

- Furthermore, while this was not explored in the current work, one could automate the process of finding the right initialisations by designing a search procedure over clause weight initialisations based on how well the entropy of the distribution over clauses is minimised by a fixed number of training steps.

Text taken directly from Evans & Grefenstette, 2018
# Baseline Tasks

| Domain   | Task            | $|P_i|$ | Recursive | $\partial$ILP | Godel  | Lukasiewicz | Max  |
|----------|-----------------|------|-----------|--------------|--------|-------------|------|
| Arithmetic | Predecessor  | 1    | No        | 100.0        | 100.0  | 100.0       | 100.0|
| Arithmetic | Even / odd      | 2    | Yes       | 100.0        | 44.0   | 52.0        | 34.0 |
| Arithmetic | Even / succ2    | 2    | Yes       | 48.5         | 28.0   | 6.0         | 20.5 |
| Arithmetic | Less than       | 1    | Yes       | 100.0        | 100.0  | 100.0       | 100.0|
| Arithmetic | Fizz            | 3    | Yes       | 10.0         | 1.5    | 0.0         | 5.5  |
| Arithmetic | Buzz            | 2    | Yes       | 14.0         | 35.0   | 3.5         | 5.5  |
| Lists    | Member          | 1    | Yes       | 100.0        | 100.0  | 100.0       | 100.0|
| Lists    | Length          | 2    | Yes       | 92.5         | 59.0   | 6.0         | 82.0 |
| Family Tree | Son            | 2    | No        | 100.0        | 94.5   | 0.0         | 99.5 |
| Family Tree | Grandparent    | 2    | No        | 96.5         | 61.0   | 0.0         | 96.5 |
| Family Tree | Husband        | 2    | No        | 100.0        | 100.0  | 100.0       | 100.0|
| Family Tree | Uncle          | 2    | No        | 70.0         | 60.5   | 0.0         | 68.0 |
| Family Tree | Relatedness    | 1    | No        | 100.0        | 100.0  | 100.0       | 100.0|
| Family Tree | Father         | 1    | No        | 100.0        | 100.0  | 100.0       | 100.0|
| Graphs   | Undirected Edge | 1    | No        | 100.0        | 100.0  | 100.0       | 100.0|
| Graphs   | Adjacent to Red | 2    | No        | 50.5         | 40.0   | 1.0         | 42.0 |
| Graphs   | Two Children    | 2    | No        | 95.0         | 74.0   | 53.0        | 95.0 |
| Graphs   | Graph Colouring | 2    | Yes       | 94.5         | 81.0   | 2.5         | 90.0 |
| Graphs   | Connectedness   | 1    | Yes       | 100.0        | 100.0  | 100.0       | 100.0|
| Graphs   | Cyclic          | 2    | Yes       | 100.0        | 100.0  | 0.0         | 100.0|

Table taken directly from Evans & Grefenstette, 2018
## Test to Handle Mislabeled Data

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>Predecessor</th>
<th>Less Than</th>
<th>Member</th>
<th>Son</th>
<th>Connectedness</th>
<th>Undirected Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.05</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.10</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.065</td>
<td>0.005</td>
<td>0.000</td>
</tr>
<tr>
<td>0.15</td>
<td>0.007</td>
<td>0.006</td>
<td>0.002</td>
<td>0.171</td>
<td>0.030</td>
<td>0.005</td>
</tr>
<tr>
<td>0.20</td>
<td>0.117</td>
<td>0.005</td>
<td>0.013</td>
<td>0.171</td>
<td>0.079</td>
<td>0.007</td>
</tr>
<tr>
<td>0.25</td>
<td>0.279</td>
<td>0.064</td>
<td>0.029</td>
<td>0.328</td>
<td>0.161</td>
<td>0.032</td>
</tr>
<tr>
<td>0.30</td>
<td>0.425</td>
<td>0.071</td>
<td>0.034</td>
<td>0.400</td>
<td>0.250</td>
<td>0.032</td>
</tr>
<tr>
<td>0.40</td>
<td>0.629</td>
<td>0.276</td>
<td>0.127</td>
<td>0.455</td>
<td>0.326</td>
<td>0.134</td>
</tr>
<tr>
<td>0.50</td>
<td>0.686</td>
<td>0.486</td>
<td>0.174</td>
<td>0.483</td>
<td>0.483</td>
<td>0.302</td>
</tr>
<tr>
<td>0.60</td>
<td>0.695</td>
<td>0.616</td>
<td>0.199</td>
<td>0.480</td>
<td>0.648</td>
<td>0.369</td>
</tr>
<tr>
<td>0.70</td>
<td>0.705</td>
<td>0.761</td>
<td>0.202</td>
<td>0.503</td>
<td>0.686</td>
<td>0.469</td>
</tr>
<tr>
<td>0.80</td>
<td>0.715</td>
<td>0.788</td>
<td>0.203</td>
<td>0.507</td>
<td>0.793</td>
<td>0.474</td>
</tr>
<tr>
<td>0.90</td>
<td>0.727</td>
<td>0.753</td>
<td>0.182</td>
<td>0.504</td>
<td>0.833</td>
<td>0.499</td>
</tr>
</tbody>
</table>
ILP with Images

- Several tests used a pre-trained CNN to answer queries about pairs of images
- Less-than example shown

(a) Proportion of successful runs
(b) Mean Error