Recent Neural-Symbolic Approaches to ILP Based on Templates

Davide Beretta[1], Stefania Monica[2], Federico Bergenti[1]

[1]Università degli Studi di Parma

[2]Università degli Studi di Modena e Reggio Emilia

EXTRAAMAS 2022

Introduction (1/3)

Deep learning has become more and more successful in the last few years, and it obtained impressive results in various research areas

The limitations of deep learning with respect to explainability and interpretability make it unsuitable for critical domains and raises several important ethical and legal concerns

Symbolic methods like *Inductive Logic Programming* can represent a way to overcome the inherent limitations of deep learning and other sub-symbolic approaches

Introduction (2/3)

Inductive Logic Programming (ILP)

An ILP task can be defined as a tuple $\langle B, P, N \rangle$ where:

- B is a set of logical rules and facts
- P is a set of positive training examples
- N is a set of negative training examples

The goal is to induce a logical program that entails all positive examples while rejecting all negative examples

ILP advantages

- Experts can inject knowledge to improve the effectiveness of learning
- ILP allows learning interpretable rules to better support XAI
- ILP methods are normally very data-efficient
- ILP supports continual and transfer learning

Introduction (3/3)

Problem

Traditional ILP methods are unable to handle noisy and non-logical data

- **Possible solution:** *Neural-symbolic* methods, which combine deep learning and ILP to obtain the best from both

Goal of this work

Survey, from the **XAI** perspective, four relevant neural-symbolic approaches to ILP that use some form of **templates** and that have been proposed in the last five years

Selected approaches are compared analyzing:

- Language
- Search method
- Recursion
- Predicate invention

δILP (1/2)

Summary

- δILP generates possible clauses from a program template
- Each pair of clauses is associated with a weight that represents the probability of being part of the generated program
- The set of weights W are obtained by training a deep neural network that tries to predict the truth value of the positive and negative examples
- The predicted truth value of each example is computed using the predetermined amount of forward chaining steps T and then applying the generated rules to background facts

Program template

Used to prune the search space, whose size grows exponentially with the number of constants and predicates

$$\Pi = \langle Pred_a, arity_a, (\pi_p^1, \pi_p^2), T \rangle$$
$$\pi_p^i = \langle v, int \rangle$$

δILP (2/2)

Given a training example (γ, λ), δILP tries to predict the probability:

$$P(\lambda|\gamma, W, \Pi, L, B)$$

where L contains constants, extensional predicates as well as the target predicate

Two major problems:

- δILP stores the weights for every pair of generated rules, which makes the method unusable in difficult tasks where the number of generated clauses is large
- The imposed restrictions on the generation of clauses makes δILP not suitable in complex tasks where a much more flexible method could produce more compact rules

NTPs (1/2)

NTPs implement a Prolog-style backward chaining algorithm that makes use of three operators: AND, OR and UNIFY. Unlike Prolog, UNIFY verifies the similarity among vector representations of symbols

Summary

- NTPs perform deduction and start with a target query
- NTPs apply the OR operator on each rule trying to unify the query with the head of the clause
- If unification succeeds, NTPs use the AND operator to jointly prove all the atoms that are part of the body of selected clause
- The AND operator performs a substitution of the variables of selected atoms, and it tries to prove them applying the OR operator again

NTPs (2/2)

Meta-rules

Meta-rules are a form of templates that allow the user to describe the structure of induced rules using a specific syntax:

$$n_1 \# m_1(X, Y) : - \# m_2(Y, X)$$

The goal of NTPs is to learn from data the representations of non-variable symbols defined in meta-rules

- Trained meta-rules can be applied to symbols with similar representations to extend the knowledge base

Consideration

NTPs seem to be more scalable than δILP, but NTPs offer less flexibility because they require to specify the structure of induced rules

ILP_{Camp} (1/2)

ILP_{Camp} is similar to NTPs but it uses forward chaining instead of backward chaining

Summary

- ILP_{Camp} allows the user to define meta-rules as NTPs
- It stores a list of facts and a set of predicate embeddings, for both background and auxiliary predicates
- It performs a predetermined number of forward chaining steps, obtaining new ground atoms
- When it finds a new ground atom, it stores the element in the list of facts, otherwise it simply updates its truth value

ILP_{Camp} (2/2)

Meta-rules

ILP_{Camp} uses a general form of meta-rules defined as:

$$F(X, Y) \Leftarrow F(X, Z), F(Z, Y)$$

Considerations

- Compared to NTPs, clause generation requires an increasing number of facts in order to perform a single step
- Compared to δILP, meta-rules represent a less flexible solution than program templates because they require a human expert to manually specify the structure of the induced rules

$Meta_{Abd}$ (1/2)

$Meta_{Abd}$ contains two major components:

- A perception module, parameterized with θ, that maps each sub-symbolic input x to its interpretation z
- A reasoner module H, which is a set of rules that, using background knowledge B and z, can be used to infer an output symbol y

Summary

- $Meta_{Abd}$ employs an Expectation-Maximization algorithm that uses a combination of induction and abduction
- The Expectation step starts inducing H, which is then used to abduce values for z
- $Meta_{Abd}$ assigns a score to each pair $H \cup z$, and it returns the pair with the highest score
- The Maximization step estimates new parameters θ using a gradient descent algorithm

$Meta_{Abd}$ (2/2)

Meta-rules

$Meta_{Abd}$ uses a form of higher-order meta-rules that can be written as:

$$metarule([P, Q], [P, A, B], [[Q, A, B]]).$$

Considerations

- Meta-rules are similar to those of ILP_{Camp}
- The literature does not document a comparison with the other methods, and it would be interesting to find how well $Meta_{Abd}$ performs
- It is not clear how well $Meta_{Abd}$ performs with the growth of the size of the training set

Language

All the discussed methods produce programs in some dialect of Datalog

- δILP learns Datalog programs by design using program templates to generate clauses
- ILP_{Camp} and NTPs learn the embeddings of rule predicates, which are not expressed as explicit Datalog programs, but they are used in the context of a Datalog knowledge base completion task
- $Meta_{Abd}$ learns programs written in a specific dialect of Datalog using an higher-order Prolog engine

Search method

The literature documents three main search methods

- *Top-down*: a general hypothesis is gradually specialized
- *Bottom-up*: a specialized hypothesis is gradually generalized
- *Meta-level*: the search problem is encoded as a meta-level logic program that is then solved by an off-the-shelf solver

Discussed methods are either top-down or meta-level:

- NTPs and ILP_{Camp} are top-down methods because they start with a single general hypothesis specified by meta-rules
- $Meta_{Abd}$ and δILP are meta-level methods, and they employ as solvers a Prolog interpreter and a neural network, respectively

Recursion

Recursion allows performing an infinite number of deduction steps with a finite logical program

- It allows avoiding the learning of a separate rule for each specific situation

All the studied methods support recursion:

- δILP provides recursion by design because it uses program templates to generate recursive clauses
- NTPs support recursion using meta-rules that manually describe how recursion is expected to be applied
- $Meta_{Abd}$ and ILP_{Camp} make use of meta-rules but they do not expect the user to manually specify relationships among the rules

Predicate invention

Background knowledge is typically provided by experts, and it is difficult to provide a comprehensive knowledge base

- Predicate invention obviates users from manually specifying the knowledge needed to solve the ILP task

Most ILP methods do not support predicate invention

- δILP supports predicate invention only partially, and it requires the user to manually specify the number and arity of auxiliary predicates
- ILP_{Camp} supports predicate invention only partially, and it allows users to specify the number of auxiliary predicates
- NTPs do not support predicate invention
- $Meta_{Abd}$ supports predicate invention, and new predicates are defined using meta-rules

Open Challenges and Conclusion

Open Challenges

- Advance over templates and meta-rules
- Design and develop a standard set of benchmarks for ILP tasks
- Develop complete and easy-to-use implementations
- Reduce the complexity of generated rules to avoid the explosion of background knowledge
- Optimize generated rules to make the learned solutions easily readable and understandable by humans

Conclusion

Neural-symbolic ILP has the potential to overcome the limits of current Machine Learning methods, allowing the development of powerful and trustable XAI

Thank you

For your attention!

Questions