Inductive Logic Programming

> Inductive logic programming (ILP) is a form of machine learning (ML). As with other forms of ML, the goal of ILP is to induce a hypothesis that generalises training examples. However, whereas most forms of ML use vectors/tensors to represent data (examples and hypotheses), ILP uses logic programs (sets of logical rules). Moreover, whereas most forms of ML learn functions, ILP learns relations.

Why ILP?

- **Data efficiency.** Many forms of ML are notorious for their inability to generalise from small numbers of training examples, notably deep learning. By contrast, ILP can induce hypotheses from small numbers of examples, often from a single example.
- **Background knowledge.** ILP learns using BK represented as a logic program. Moreover, because hypotheses are symbolic, hypotheses can be added the to BK, and thus ILP systems naturally support lifelong and transfer learning.
- **Expressivity.** Because of the expressivity of logic programs, ILP can learn complex relational theories. Because of the symbolic nature of logic programs, ILP can reason about hypotheses, which allows it to learn optimal programs.
- **Explainability.** Because of logic's similarity to natural language, logic programs can be easily read by humans, which is crucial for explainable AI.

Recent Advances

- Search: Meta-level
- Recursion: Yes
- Predicate Invention: Limited
- Hypotheses: Higher-order; ASP
- Optimality: Yes
- Technology: Prolog; ASP; NNs

Review

- **Search.** The fundamental ILP problem is to efficiently search a large hypothesis space. Most older ILP approaches search in either a top-down or bottom-up fashion. A third new search approach has recently emerged called meta-level ILP.
  - **Top-down** approaches start with a general hypothesis and then specialise it.
  - **Bottom-up** approaches start with the examples and generalise them.
  - **Meta-level.** (Most) approaches encode the ILP problem as a program that reasons about programs.
- **Recursion.** Learning recursive programs has long been considered a difficult problem for ILP. The power of recursion is that an infinite number of computations can be described by a finite recursive program.

- Interest in recursion has resurged with the introduction of meta-interpretive learning (MIL) and the MIL system Metagol. The key idea of MIL is to use metarules, or program templates, to restrict the form of inducible programs, and thus the hypothesis space. A metarule is a higher-order clause. Following MIL, many meta-level ILP systems can learn recursive programs. With recursion, ILP systems can now generalise from small numbers of examples, often a single example. Moreover, the ability to learn recursive programs has opened up ILP to new application areas.

- **Predicate invention.** A key characteristic of ILP is the use of BK. BK is similar to features used in most forms of ML. However, whereas features are tables, BK contains facts and rules (extensional and intensional definitions) in the form of a logic program.

  - Rather than expecting a user to provide all the necessary BK, the goal of predicate invention (PI) is for an ILP system to automatically invent new auxiliary predicate symbols. Whilst PI has attracted interest since the beginnings of ILP, and has subsequently been repeatedly stated as a major challenge, most ILP systems do not support it.
  - Several PI approaches try to address this challenge: Placeholders, Metarules, Pre/post-processing, Lifelong Learning.
  - The aforementioned techniques have improved the ability of ILP to invent high-level concepts. However, PI is still difficult and there are many challenges to overcome. The challenges are that (i) many systems struggle to perform PI at all, and (ii) those that do support PI mostly need much user-guidance, metarules to restrict the space of invented symbols or that a user specifies the arity and argument types of invented symbols.
  - ILP systems have traditionally induced definite and normal logic programs, typically represented as Prolog programs. A recent development has been to use different hypothesis representations.
    * **Datalog** is a syntactical subset of Prolog which disallows complex terms as arguments of predicates and imposes restrictions on the use of negation. The general motivation for reducing the expressivity of the representation language from Prolog to Datalog is to allow the problem to be encoded as a satisfiability problem, particularly to leverage recent developments in SAT and SMT.
    * **Answer Set Programming** (ASP) is a logic programming paradigm based on the stable model semantics of normal logic programs that can be implemented using the latest advances in SAT solving technology.
      · When learning Prolog programs, the procedural aspect of SLD-resolution must be taken into account. By contrast, as ASP is a truly declarative language, no such consideration need be taken into account when learning ASP programs. Compared to Datalog and Prolog, ASP supports addition language constructs, such as disjunction in the head of a clause, choice rules, and hard and weak constraints.
      · A key difference between ASP and Prolog is semantics. A definite logic program has only one model (the least Herbrand model). By contrast, an ASP program can have one, many, or even no stable models (answer sets). Due to its non-monotonicity, ASP is particularly useful for expressing common-sense reasoning.
      · Approaches to learning ASP programs can mostly be divided into two categories: *brave learners*, which aim to learn a program such that at least one answer set covers the examples, and

cautious learners, which aim to find a program which covers the examples in all answer sets.

* **Higher-order programs** where predicate symbols can be used as terms.
* **Probabilistic logic programs.** A major limitation of logical representations, such as Prolog and its derivatives, is the implicit assumption that the BK is perfect. This assumption is problematic if data is noisy, which is often the case.
    - Integrating probabilistic reasoning into logical representations is a principled way to handle such uncertainty in data. This integration is the focus of statistical relational artificial intelligence (StarAI). In essence, StarAI hypothesis representations extend BK with probabilities or weights indicating the degree of confidence in the correctness of parts of BK. Generally, StarAI techniques can be divided in two groups: *distribution representations* and *maximum entropy* approaches.
    - **Distribution semantics** approaches explicitly annotate uncertainties in BK. To allow such annotation, they extend Prolog with two primitives for stochastic execution: probabilistic facts and annotated disjunctions. Probabilistic facts are the most basic stochastic primitive and they take the form of logical facts labelled with a probability p. Each probabilistic fact represents a Boolean random variable that is true with probability p and false with probability $1 - p$. Whereas probabilistic facts introduce non-deterministic behaviour on the level of facts, annotated disjunctions introduce non-determinism on the level of clauses. Annotated disjunctions allow for multiple literals in the head, where only one of the head literals can be true at a time.
    - **Maximum entropy** approaches annotate uncertainties only at the level of a logical theory. That is, they assume that the predicates in the BK are labelled as either true or false, but the label may be incorrect. These approaches are not based on logic programming, but rather on first-order logic. Consequently, the underlying semantics are different: rather than consider proofs, these approaches consider models or groundings of a theory. This difference primarily changes what uncertainties represent. For instance, Markov Logic Networks (MLN) represent programs as a set of weighted clauses. The weights in MLN do not correspond to probabilities of a formula being true but, intuitively, to a log odds between a possible world (an interpretation) where the clause is true and a world where the clause is false.
    - The techniques from learning such probabilistic programs are typically direct extensions of ILP techniques.

- **Optimality.** There are often multiple (sometimes infinitely many) hypotheses that explain the data. Deciding which hypothesis to choose has long been a difficult problem.
    - Older ILP systems were not guaranteed to induce optimal programs, where optimal typically means with respect to the size of the induced program or the coverage of examples. A key reason for this limitation was that most search techniques learned a single clause at a time, leading to the construction of sub-programs which were sub-optimal in terms of program size and coverage.
    - Newer ILP systems try to address this limitation. As with the ability to learn recursive programs, the main development is to take a global view of the induction task by using meta-level search techniques. In other words, rather than induce a single clause at a time from a single example, the idea is to induce

multiple clauses from multiple examples.
  - The ability to learn optimal programs opens up ILP to new problems. For instance, learning efficient logic programs has long been considered a difficult problem in ILP, mainly because there is no declarative difference between an efficient program and an inefficient program.

- **Technologies.** Older ILP systems mostly use Prolog for reasoning. Recent work considers using different technologies.
  - **Constraint satisfaction and satisfiability.** There have been tremendous recent advances in SAT.
    * To leverage these advances, much recent work in ILP uses related techniques, notably ASP. The main motivations for using ASP are to leverage (i) the language benefits of ASP, and (ii) the efficiency and optimisation techniques of modern ASP solvers, which supports conflict propagation and learning.
    * With similar motivations, other approaches encode the ILP problem as SAT or SMT problems. These approaches have been shown able to reduce learning times compared to standard Prolog-based approaches.
    * However, some unresolved issues remain. A key issue is that most approaches encode an ILP problem as a single (often very large) satisfiability problem. These approaches therefore often struggle to scale to very large problems, although preliminary work attempts to tackle this issue.
  - **Neural Networks.** With the rise of deep learning, several approaches have explored using gradient-based methods to learn logic programs. These approaches all replace discrete logical reasoning with a relaxed version that yields continuous values reflecting the confidence of the conclusion.

- **Applications.**
  - **Scientific discovery.** Perhaps the most prominent application of ILP is in scientific discovery: identify and predict ligands (substructures responsible for medical activity) and infer missing pathways in protein signalling networks; ecology.
  - **Program analysis.** learning SQL queries; programming language semantics, and code search.
  - **Robotics.** Robotics applications often require incorporating domain knowledge or imposing certain requirements on the learnt programs.
  - **Games.** Inducing game rules has a long history in ILP, where chess has often been the focus
  - **Data curation and transformation.** Another successful application of ILP is in data curation and transformation, which is again largely because ILP can learn executable programs. There is much interest in this topic, largely due to success in synthesising programs for end-user problems, such as string transformations. Other transformation tasks include extracting values from semi-structured data (e.g. XML files or medical records), extracting relations from ecological papers, and spreadsheet manipulation.
  - **Learning from trajectories.** Learning from interpretation transitions (LFIT) automatically constructs a model of the dynamics of a system from the observation of its state transitions. LFIT has been applied to learn biological models, like Boolean Networks, under several semantics: memory-less deterministic systems, and their multi-valued extensions. The Apperception Engine explain sequential data, such as cellular automata traces, rhythms and simple nursery tunes, image occlusion tasks, game dynamics, and sequence induction intelligence tests. Surprisingly, can achieve human-level performance on the

sequence induction intelligence tests in the zero-shot setting (without having been trained on lots of other examples of such tests, and without hand-engineered knowledge of the particular setting). At a high level, these systems take the unique selling point of ILP systems (the ability to strongly generalise from a handful of data), and apply it to the self-supervised setting, producing an explicit human-readable theory that explains the observed state transitions.

- **Limitations and future research.**
  - **Better systems.** A problem with ILP is the lack of well engineered tools. They state that whilst over 100 ILP systems have been built, less than a handful of systems can be meaningfully used by ILP researchers. By contrast, driven by industry, other forms of ML now have reliable and well-maintained implementations, which has helped drive research. A frustrating issue with ILP systems is that they use many different language biases or even different syntax for the same biases. *For ILP to be more widely adopted both inside and outside of academia, we must develop more standardised, user-friendly, and better-engineered tools.*
  - **Language biases.** One major issue with ILP is choosing an appropriate language bias. Even for ILP experts, determining a suitable language bias is often a frustrating and time-consuming process. We think the need for an almost perfect language bias is severely holding back ILP from being widely adopted. *We think that an important direction for future work in ILP is to develop techniques for automatically identifying suitable language biases.* This area of research is largely under-researched.
  - **Better datasets.** Interesting problems, alongside usable systems, drive research and attract interest in a research field. This relationship is most evident in the deep learning community which has, over a decade, grown into the largest AI community. This community growth has been supported by the constant introduction of new problems, datasets, and well-engineered tools. ILP has, unfortunately, failed to deliver on this front: most research is still evaluated on 20-year old datasets. Most new datasets that have been introduced often come from toy domains and are designed to test specific properties of the introduced technique. *We think that the ILP community should learn from the experiences of other AI communities and put significant efforts into developing datasets that identify limitations of existing methods as well as showcase potential applications of ILP.*
  - **Relevance.** New methods for predicate invention have improved the abilities of ILP systems to learn large programs. Moreover, these techniques raise the potential for ILP to be used in lifelong learning settings. However, inventing and acquiring new BK could lead to a problem of too much BK, which can overwhelm an ILP system. On this issue, a key under-explored topic is that of relevancy. *Given a new induction problem with large amounts of BK, how does an ILP system decide which BK is relevant?* One emerging technique is to train a neural network to score how relevant programs are in the BK and to then only use BK with the highest score to learn programs. Without efficient methods of relevance identification, it is unclear how efficient lifelong learning can be achieved.
  - **Handling mislabelled and ambiguous data.** A major open question in ILP is how best to handle noisy and ambiguous data. Neural ILP systems are designed from the start to robustly handle mislabelled data. Although there has been work in recent years on designing ILP systems that can handle noisy mislabelled data, there is much less work on the even harder and more fundamental problem of designing ILP systems that can handle raw ambiguous data. ILP systems typically assume that the input

has already been preprocessed into symbolic declarative form (typically, a set of ground atoms representing positive and negative examples). But real-world input does not arrive in symbolic form. *For ILP systems to be widely applicable in the real world, they need to be redesigned so they can handle raw ambiguous input from the outset.*

- **Probabilistic ILP.** Real-world data is often noisy and uncertain. Extending ILP to deal with such uncertainty substantially broadens its applicability. While StarAI is receiving growing attention, **learning probabilistic programs from data is still largely under-investigated due to the complexity of joint probabilistic and logical inference.** When working with probabilistic programs, we are interested in the probability that a program covers an example, not only whether the program covers the example. Consequently, probabilistic programs need to compute all possible derivations of an example, not just a single one. Despite added complexity, probabilistic ILP opens many new challenges. Most of the existing work on probabilistic ILP considers the minimal extension of ILP to the probabilistic setting, by assuming that either (i) BK facts are uncertain, or (ii) that learned clauses need to model uncertainty. **These assumptions make it possible to separate structure from uncertainty and simply reuse existing ILP techniques.** Following this minimal extension, the existing work focuses on discriminative learning in which the goal is to learn a program for a single target relation. However, a grand challenge in probabilistic programming is generative learning. That is, learning a program describing a generative process behind the data, not a single target relation. **Learning generative programs is a significantly more challenging problem, which has received very little attention in probabilistic ILP.**
- **Explainability.** Explainability is one of the claimed advantages of a symbolic representation. Recent work evaluates the comprehensibility of ILP hypotheses using Michie's framework of ultra-strong machine learning, where a learned hypothesis is expected to not only be accurate but to also demonstrably improve the performance of a human being provided with the learned hypothesis. [Some work] empirically demonstrate improved human understanding directly through learned hypotheses. *However, more work is required to better understand the conditions under which this can be achieved, especially given the rise of PI.*

Bibliography

1. Inductive logic programming at 30