[DRAFT] Comparing Inductive Logic Programming & Deep Learning

Gavin Leech[1], Javier Prieto[2], Nandi Schoots[3]

March 2020

Abstract

Inductive Logic Programming (ILP) is a collection of declarative ML methods which use logic programs to represent both inputs and the output model, using inductive search to find the latter. Three benefits drew our attention: ILP's sample efficiency, its terse and high-level model outputs, and its potential for exact model surgery. These are notable as a *disjoint set* of the benefits of deep learning (DL). This suggests potential gains from hybrid ILP-DL systems. We give an introduction to ILP, compare its formal and empirical properties to those of deep learning, and inspect potential improvements.

1 Introduction

Deep learning (DL) has rapidly increased in capability, without commensurate progress in its interpretability, sample efficiency, or verifiability [1], [2]. Are there ML paradigms that can substitute or complement DL, to make up for this?

We consider one candidate: Inductive Logic Programming (ILP). ILP is a subfield of ML for learning from examples (E) and suitably encoded human "background knowledge" (B), using logic programs to represent both inputs (E, B) and the output model h. We included a short introduction to ILP in section 2.

[1]University of Bristol
[2]Universidad Autónoma de Madrid
[3]King's College London and Imperial College London

This research was supported by UKRI studentship funding (EP/S022937/1) and the AI Safety Research Programme, `https://aisrp.org`.

ILP was initially motivated by the promise of learning from structured data (for instance, recursive structures), and of better knowledge representation, beyond the 'propositional' context of most other ML work [3]. The resulting paradigm has many interesting properties. For example, ILP yields relatively short, human-readable models, and is sample efficient.

We investigate the safety properties in section 3 and the capabilities in section 4. In particular, we discuss: its terse and high-level model outputs (Section 3.4.1), its potential for predictable model surgery (Section 4.2), and its impressive sample efficiency (Section 4.2.4).

The upsides of ILP are in some sense a *complement* of the benefits of deep learning, which is instead known for its tolerance of unstructured, noisy, and ambiguous data, and its learning of hierarchical feature representations. When we consider the relative neglect of ILP this suggests great gains from ILP and hybrid ILP-DL systems. As a suggestive bound on the ratio of investment in ILP vs DL, compare the 130 researchers (worldwide) listed on the ILP community hub [4], to the 420 researchers at a *single* DL/RL lab, Berkeley AI Research [5]

In this work, we focus on state-of-the-art ILP systems; for a more complete view, see Sammut (1993) [6] or Muggleton et al (2011) [7]. Recent developments are promising: work on learning higher-order programs dramatically improves on previous first-order logic (FOL) systems [8]; probabilistic forms of ILP attempt to handle noise and ambiguity in the data [9]; and there is a plethora of attempts at neural-symbolic or differential versions of ILP (see Section 5.2). For instance, the ∂ILP system from researchers at Deepmind used a differentiable ILP system with the aim of integrating inductive search with DL modules [10].

In section 5, we end with a speculative take on the potential of ILP for the future of AI.

2 Background

As the name suggests, ILP is inductive logic plus logic programming: it constructs logic program generalisations from logic program examples. (For introductions to each technique, see [11] and [12].) Both the data and the resulting hypothesis are represented in formal logic, usually of the first-order or second-order. For computability reasons, systems use subsets of first-order logic, often the definite clausal logic (see Section 4.2.2). (For a complete proof-theoretic, model-theoretic, and philosophical justification of ILP, see Muggleton and de Raedt (1994) [13].)

The output of a call to an ILP system (what is induced by the learning algorithm) takes several names: a 'theory', a 'hypothesis', a 'program', a 'concept',

$$\overbrace{\forall X \ : \ (\overbrace{a \wedge b}^{\text{Clause}}) \vee (\overbrace{\neg c}^{\text{Literal}} \wedge \overbrace{X}^{\text{Variable}})}^{\text{Formula}}$$

Figure 1: Structure of a single logic program. A ‘formula’ is either an example or an output hypothesis.

or a ‘model’ (in the machine learning sense, and not the logical sense of a truth-value interpretation).

We say that ILP is a collection of methods, rather than one technique or even family of algorithms, due to the many systems not based on Prolog-like inference, and the many nonsymbolic ILP systems. Our working criterion is just that the output of an ‘ILP’ system should be a logic program.

In the classic setting, the examples E are labelled with a binary class: positive examples (E^+) and negative examples (E^-). An ILP system searches a hypothesis space $\mathcal{H}$ until a program h is found such that $B \wedge h \models E^+$, and such that $\forall e \in E^- \ B \wedge h \not\models e$. In practice this is weakened in two ways: firstly by heuristic scoring, so that *most* positives and few negatives are covered by h; and secondly by using θ-subsumption rather than normal (undecidable) FOL entailment (see Section 4.2.2). h is then a relational description, in terms of B, of some concept common to the positive examples and absent from the negatives.

The normal ILP setting assumes that atoms are either true or false, and that hypotheses have a binary domain. Thus the first ILP designs produced only binary classifiers. But ‘upgraded’ (relational)[14] forms of many propositional ML techniques have been developed: multi-class classification [15], regression with decision trees [16], clustering [17], and even visual object classification [18]. This bivalence also entails the inability of early, exact ILP systems to handle ambiguous data.

We can view ILP as a search of the ‘subsumption lattice’, the graph that results from partially ordering hypotheses in $\mathcal{H}$ from most general (true $\to E^+$) to most specific (the bottom clause $\perp$, a conjunction of evaluated predicates), see Figure 1.

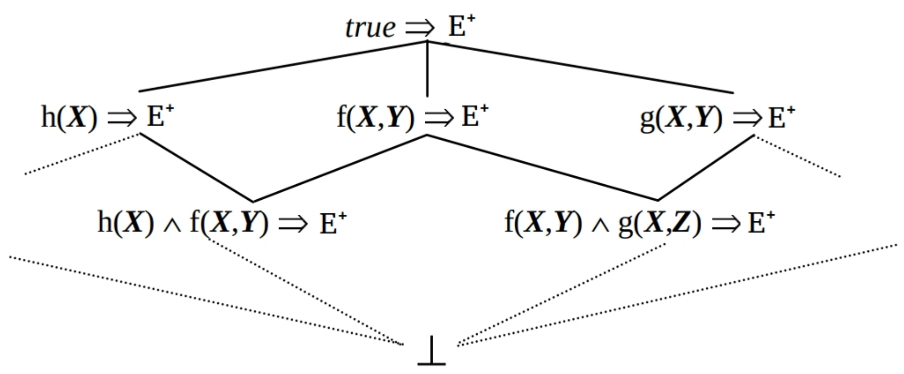

Figure 2: Example of a subsumption lattice to search. Adapted from [19]

There are, then, two approaches to hypothesis discovery in ILP: 'bottom-up' search, which proceeds from an initially long clause (the features of individual examples), finds a specific clause to generalise from, and drops or abstracts away literals until a minimally general hypothesis that covers E^+ is found. This specific-to-general search direction is the default approach in the classic ILP systems Progol and Aleph (see below). The 'top-down' approach proceeds from a short clause (for instance, the empty implication `true`), and adds literals to it until the expression becomes too specific. This generally involves generating candidate clauses from a language template, then testing these clauses against E, branching to other paths through the lattice when violations are found. This general-to-specific approach is used in Metagol and ∂ILP.

The expressive power of (even subsets of) first-order logic leads to ILP's computational complexity: the resulting combinatorial search over large discrete spaces is a notably difficult problem: in fact, it is in the complexity class NP^{NP} (see Section 4.2.3). As a result, various forms of heuristic scoring are used to guide the search.

Table 1 is an attempt to understand the abstract learning properties of ILP and DL. For instance, we can draw an analogy between the 'program template' that constrains ILP's hypothesis space and the architecture of a neural network; both constrain the hypothesis space and, until recently, both have been entirely handcrafted, though recent results in neural architecture search promises partial automation of this bias provision [20]. We draw the division of inductive bias into language bias (hypothesis space restriction), procedural bias (how the search is ordered; also called 'search bias'), and simplicity bias (how overfitting is prevented) from Witten et al [21].

Both DL and ILP programs depend on strong inductive bias to find a good model. In DL inductive bias takes the form of architecture search. The archi-

Learning property	**ILP**	**DL**
Language bias	Program templates	NN architecture
Procedural bias	Various: classic search algos	N/A
Simplicity bias	Bound on program length	Controversial [22][23]
Search procedure	Local search for subsumption[?]	Local search over gradients
Automated language bias?	No [10]	Sometimes [20]

Table 1: Some abstract properties of the two approaches

tecture of Neural network architecture search has been successful in some cases [20]. However, program template search is still an open problem: early experiments in brute-forcing templates gave understandably poor results, and the only other related work to date involves the intelligent *selection* or distillation of metarules, rather than generation [24].

2.1 Classifying ILP Systems

There exist dozens of ILP systems [25]. To keep the present survey manageable, we study the following exemplars: FOIL (1990) [26], Progol (1995) [27], Aleph (1999) [28], Metagol (2013) [29], ∂ILP (2018) [10]. We classify ILP systems using the following dimensions:

1. *Order of hypotheses*: does it allow first-order logic or higher-order logics in the representation of inputs, intermediates, and outputs?

2. *Target language*: Almost all ILP systems induce Prolog programs; however recent systems use other target languages, for instance Datalog (less expressive than Prolog) or answer-set programming[30] (more expressive).

3. *Search strategy*: whether the search is conducted top-down (that is, from general to specific) or bottom-up (starting with an example and generalising it), or whether both are used (as in 'theory revision').

4. *Exact or probabilistic search*: does the search include stochastic steps?

5. *Noise handling*: how are mislabeled examples or other corrupt data-points handled? An implementation detail, this can involve restricting specialisation in top-down search; allowing some negatives to be covered by a clause; or by using a neural network to preprocess data.

6. *Inference engine*: Almost all ILP systems are meta-interpreters running on Prolog. More recent systems attempt to replace symbolic inference with latent embeddings in a structured neural network or some other differentiable structure, for instance the differentiable deduction (∂D) of ∂ILP.

7. *Predicate invention*: can the system induce new background assumptions during learning?

System	**Order**	**Target**	**Search**	**Exact**	**Noise**	**Engine**	**Invent**
FOIL	First	Prolog	Top-down	Yes	None	Prolog	No
Progol	First	Prolog	Top-down	Yes	Bounded	Prolog	No
Aleph	First	Prolog	Bottom-up	Both	Bounded	Prolog	No
Metagol	Higher	Datalog/ASP	Top-down	Yes	Scoring	Prolog	Yes
∂ILP	First	Datalog	Top-down	No	ANN	∂D	Yes

Table 2: Key dimensions of key ILP systems

ILP systems differ along many other dimensions, but for our purposes the above are most informative. Other design decisions can be found in the appendix, Section 7. Code listings for three ILP algorithms (an abstraction over all such systems; the top-down search of FOIL; and the bottom-up search of Aleph) can be found in Section 8.

3 Comparing safety properties of DL and ILP

Consider a machine learning system 'safe' when we can ensure it behaves correctly according to some specification. [cite Russell] An ML system can behave incorrectly in many ways, We take as a premise that some ML approaches make it easier to mitigate failure modes. The present study focuses on ILP as a candidate for 'safe' machine learning.

For example, two ways in which ML systems can behave incorrectly include: the model behaves incorrectly because we did not specify correctly what it should have learned, e.g. in DL if we use a loss function that isn't tied to what we want the system to learn, then it will learn something we don't want; the model seems to behave correctly, but we have little insight into its decision mechanism and so can't predict how it will behave on data out of the learning distribution.

Ideally, we use a model which is provably safe and its decision mechanism can be satisfactory explained to people with different levels of AI education, e.g. to an ML researcher, a marketing specialist and a CEO.

Contrary to DL models, ILP models (output hypotheses/theories) are represented in an explicit language, which intuitively makes it easier to analyse and reason about the models and predict their behaviours.

In the sections below we explore the following safety properties: interpretability, predictable model surgery, value loading and interactivity.

3.1 Interpretability

To explain an ML system's performance and to detect undesirable states before they manifest in behaviour, we need the output model and the training procedure to be interpretable. We follow Muggleton [31] in operationalising a model's (program's) interpretability as "the mean accuracy with which after brief study and without further sight they can use P to classify new material" [31].

We can also understand it by its edge cases. A maximally interpretable model might be the univariate linear regression: it is a global model completely specified by two parameters, each of which has a natural interpretation which does not vary as the values scale, and each can be understood independent of the other. Conversely, an example of a minimally interpretable model is some very large neural network: the millions of parameters form a distributed representation of a nonlinear function, such that it is generally uninformative to reason about individual activations, and such that, until very recently, it has not been possible to identify the relevant subnetworks for the purposes of explaining a given decision of the whole network [32].

The notable interpretability researcher Christopher Olah notes four safety benefits of interpretable systems [33]:

1. *Test-time prevention of unsafe behaviour.* If we can audit a model thoroughly - that is, if we really understand, on the level of composition of basic concepts, what our model has actually learned (and not just what the evaluation metrics tell us about its outputs), then we can reliably prevent the deployment of unsafe systems of certain kinds (those without adversarial incentives to foil the interpreter).

2. *Guided search through AI space.* If we understand why our models work, then we should thereby also understand their failure modes. The intuition is that this would allow for more control over the design, such that we could avoid building unsafe systems in the first place. (Further, if this led to differential improvements in more transparent systems, then further interpretability improvements would be greatly incentivised.

3. *Improved feedback at train-time.* Human feedback could become a rich source of inductive bias and safety constraints, any part of the learning problem which is hard to specify explicitly. If we included a transparency term in the objective of a system, then it would be easier to alter a training process to respect constraints, and also perhaps easier to track learning at a high-level in real time.

4. *Microscope AI: advanced model extraction.* Many safety problems could be obviated entirely if the resulting system was not deployed. Instead, it could be that very advanced interpretability tools extract the models learned by very advanced ML systems, and then we (or our ordinary software) use

> these insights for improved actions. The purpose of such a 'microscope' AI would be to learn, and then expose this implicit knowledge to us.

ILP may be a natural fit for safe learning: often not just the output model, but also the learning process takes place at a relatively high-level (that is, involves humanlike reasoning about coverage, general cases, and special cases). [clarify.] A small usability study of ILP in terms of its interpretability by nonexpert humans found that TODO [31].

Rule-based models are receiving attention in another part of AI ethics: work by Cynthia Rudin and collaborators shows that rule-based models can perform well while guaranteeing properties like demographic or procedural fairness [34], [35], [36]. Related work includes explaining large ensembles of trees by locally approximating them with logic programs [37]

3.2 Deterministic model surgery

The symbolic semantics of learned ILP models (simply that of first-order or second-order logic) make them easier to manipulate than DL models.

After training a neural network we have a model that acts almost like a blackbox. This blackbox is not easy to manipulate. Through active learning we can update the model and we could for example add a module that deals with exeptions to the general training data. However, active learning does not give us much control over exactly how the blackbox changes. Adding a module isn't very elegant, because there is no synergy between the learned blackbox and the module. Because the blackbox is difficult to interpret, researchers do not fully understand what function the network has learned and so are not empowered to enhance it. They can choose to override the models output with a different module or have faith in the model, but there is no synergystic combination of learned model and human insight.

A model learned using ILP on the other hand is represented in interpretable logic, which makes it easier to manipulate and tweak while retaining the strengths of the original model.

3.3 Robustness to input change

The exact semantics of ILP (simply that of FOL or SOL) should mean that the changes in coverage from altering one atom of a model are fully predictable. Adding a negation somewhere could completely change the output.

ILP is invariant to how the problem is represented. (Semantically the same examples lead to the same output.) Changing the problem (background and examples), will usually change the output program

Deep learning is less sensitive to small changes (or noise) in the input.

3.4 Verification of specifications

If you have an output model and want to verify whether it satisfies some specification, then this is in theory NP-hard for both a neural network and a program outputted by ILP. In practice, it can be doable.

To cite [38], "Deep neural networks are large, non-linear, and non-convex, and verifying even simple properties about them is an NP-complete problem". State of the art complete solvers can in practice verify properties of networks with thousands of nodes (but do timeout a lot for larger networks) [39]. Incomplete methods can verify many properties of networks with 100 000 ReLU nodes [40], [41]. (The smallest networks that achieve relatively good results on the CIFAR10 dataset have 50-100k nodes.) Networks can be trained to ease verification [42].

Verifying properties of the output of ILP system is in practice usually easy, but can be NP hard. Suppose you have a propositional theory and verify whether this satisfies the specification False. This is equivalent to solving Boolean satisfiability, so ILP specification verification is at least NP hard. It is in general uncomputable with Prolog theories, and hence not in NP (and therefore also not NP complete). For specific fragments of Prolog it may be different. For example, with Datalog it can be solved in EXP time.

It is usually easy to verify whether an ILP model satisfies a specification [CITATION], whereas this can be quite difficult with neural networks. Specific methods to verify whether a specification holds include ...

Often, however, verification of specifications of the output program are not necessary, because we can just add the specifications to the background before we even start learning the output program.

3.5 Model specification and value loading

If we know what specifications we want to hold for our output model, then in ILP we can make sure the search only finds programs that satisfy these specifications. If you want to guarantee that the learned model satisfies some specification ϕ, then all you have to do is add ϕ to the training data/examples.

Whether a program that ILP outputs satisfies all desired properties depends on the quality of the input. In cases where the input does give a complete representation of the desired properties, if ILP outputs a program, it is correct by construction. The positive and negative input examples (and background) could be insufficient, in that they do not completely capture all the properties that one cares about. Furthermore, if noise exists in data then the hypothesis is not guaranteed to be correct even for an exact ILP system.

Unfortunately, not all ILP systems are complete, i.e. a solution may exist even though the system doesn't find one. See Section 7.

Norms and values are a particular type of specifications. If we want to load values into the output program, then we can just add these to the background knowledge. Naively, ILP's ability to represent human knowledge is helpful - say for simple, queryable value loading. But it is unwieldy to encode context in FOL clauses, and unfortunately much of value is contextual.

Specifying a model in DL is very difficult, and computationally expensive (often prohibitively) [CITATION].
Look at neural networks with constraints. Also consider penalizing heavily for something not fitting a value.

If you have specific constraints, e.g. regulations.

3.6 Control over inductive bias

Correct behaviour is generally underdetermined by the training data, so in order to get the right behaviour you need the right inductive bias. This makes it attractive to have leverage over the training algorithm through control over the inductive bias.

Neural networks have a broad bias towards simplicity, but it is unclear why, and it would not be straightforward to change it.

ILP templates (the search order) are easy to specify, so you have control over the inductive bias. There is a version of Metagol that searches for fast programs rather than short ones.

4 Comparing the Capabilities of DL and ILP

The safety properties mentioned in the previous section are only relevant if its capabilities can compete with deep learning.

4.1 State of the art empirical performance

In [10], ∂ILP is introduced and its performance is compared with Metagol on 20 discrete error-free symbolic tasks. In the 3 tasks where Metagol did not manage to find a solution it had gotten stuck in an infinite loop due to the presence of a recursive metarule interacting with a cycle in the data [10]. On the other hand, ∂ILP is limited in that it requires a prohibitively large amount of memory, because of which problems that require ternary predicates are currently outside the scope of ∂ILP [10].

The 90s saw many applications of ILP in chemistry, drug design, and biology [43], [44], [45], [46]. Contributions from the ILP community to these important applications are ongoing [47], [48], [49], [50], but the size of the field no longer compares to the counterpart effort using deep learning [DL citations]. In the early 2000's ILP had success in the Comparative Assessment of protein Structure Prediction (CASP) competition [51], [52], [53], but these days most submissions use DL. In 2018, DeepMind's AlphaFold booked success in the competion [54].

When evaluating the few direct benchmarks solved by both ILP and DL, it is important to recall the large difference in research investment between the two programmes: what may seem like an unbeatable lead in capabilities may in fact be an artefact of this difference.

4.2 Theoretical Bounds on Performance

4.2.1 Expressivity

"The class of dyadic logic programs with one function symbol has Universal Turing Machine (UTM) expressivity" [55]
The expressiveness of a formal language is a measure of its ability to distinguish structures. We can only compare the expressiveness of two languages if we are interpreting the same class of structures with them.

Is there a function which one can represent which the other cannot?

Which system is more expressive? There are at least two contradictory directions:
1) The logical hierarchy

$$ILP \rightarrow \text{FOL}$$

$$DL \rightarrow \text{propositional logic}$$

$$\text{Expressivity(FOL)} > \text{Expressivity(prop)}$$

$$\text{Expressivity(ILP)} > \text{Expressivity(DL)}$$

However, natural neural networks have proven themselves to be very powerful in the past. Humans seem an existence proof that there are neural networks that can learn logic.

2) The numerical hierarchy

ILP deals with discrete data;
DL deals with continuous data Continuous data > discrete data

$$Expressivity(ILP) > Expressivity(DL)$$

3) Claim: logic $\subset$ probability
Dubious: inference rules aren't the same.

Theoretical Expressivity

We first compare the two paradigms on what the algorithms could in theory learn, regardless of how inefficient it would be to learn this.

You could – in theory – encode tensors and tensor operations (such as activation functions) as logical sentences.

Likewise, you could in theory enumerate all logical sentences up to length n and then encode them as a gigantic one-hot vector. (You could also come up with a more elegant and shorter description.) A 2-layer neural network that is wide enough can approximate all possible functions from this vector to $\{0, 1\}$ arbitrarily closely. Neural networks are functions from R^n to R^m. Neural networks can approximate any continuous function.

Practical Expressivity

We now compare ILP and DL on what they can be used for in practice.

In ILP it is generally awkward to work with data that is non-categorical.

On the other hand it is difficult to express "self-referential" functions using a neural network. For example, the following are all hard to learn for a neural network: an identity function; recognizing palindromes; and Fizzbuzz. Neural networks are a propositional method, it has no quantifiers and no equality constraints. This makes it hard to make recursion work.

4.2.2 Computability

As noted in Section 2, ILP is a search for clauses that entail given positive examples, but entailment in general is semi-decidable [CITATION NEEDED]. This means that whenever $H_1 \vdash H_2$, there is an algorithm to prove it, but no algorithm exists that can prove $H_1 \not\vdash H_2$ when $H_1 \not\vdash H_2$. If we restrict ourselves to Horn clauses in function-free FOL we can even give a sharper characterisation: finding out whether $H_1 \vdash H_2$ is undecidable if H_1 has more than one atom in its body [56]. To circumvent this issue, we must provide a suitable notion of provability ($\models$) to the ILP algorithm.

A reasonable candidate is implication, which is both correct

$$(A \models B) \Rightarrow (A \vdash B)$$

and complete

$$(A \vdash B) \Rightarrow (A \models B).$$

However, even in this setting, the problem remains undecidable in general and even for smaller sets of FOL clauses (see [57], theorems 2 through 6).

A better-behaved alternative to implication is θ-subsumption. Given a variable substitution θ, we say H θ-subsumes e, and write $H \leq_\theta e$, if the literals in $H\theta$ are a subset of those in e. For example, if $H = P(X, Y)$ and $e = P(a, b)$ we can show $H \leq_\theta e$ simply taking $\theta = \{X/a, Y/b\}$.

It can be shown that θ-subsumption is correct, but not complete, since it is decidable [CITATION NEEDED]. However, completeness is guaranteed if we restrict ourselves to non self-resolving clauses, which means we cannot consider predicates like married(X,Y) :- married(Y,X)[57]. [this is incomplete, see theorem 10 in [57]]. Given these results, we restrict the following sections to ILP problems using θ-subsumption.

4.2.3 Complexity

We now turn to complexity results. A grain of salt first: Complexity theory often underestimates practical progress in concrete AI and optimisation systems. Consider the Travelling Salesman problem, infamously NP-hard, is routinely solved (at least approximately) by thousands of companies to optimise their logistics. This should make us suspicious of the applicability of general complexity results to any particular setting.
A more recent example: neural networks trained with reinforcement learning have made a dent in PSPACE-hard problems like playing Go [58], and more generally worst-case theories like Rademacher complexity overestimate the generalisation error of neural networks [59]. Some possible explanations of this apparent discrepancy are:

1. *Giving up worst-case performance*: Worst-case complexity is not the same as average-case complexity [60]. Since, by definition, the algorithm will be dealing with average instances most of the time, worst-case performance may not be of practical importance. (However, this complacent view may make a system vulnerable to adversarial attack.)

2. *Giving up optimality*: If we assume diminishing returns to optimisation, a suboptimal solution within a fixed margin (ϵ) of the optimal one may be exponentially ($\exp 1/\epsilon$) faster to find.

3. *Giving up correctness*: Randomisation can reduce hardness significantly [CITATION NEEDED]. The cost of giving up correctness can be offset in some cases by several independent runs of the algorithm that make the probability of an error vanish exponentially with the number of runs.

4. *Giving up generalisation*: A narrower algorithm may be faster and still useful in most cases [CITATION + examples].

However, complexity theory is still the best theoretical tool available to study the efficiency of a given algorithm or hardness of a given problem, and in particular to compare . In this, we will follow [57] and [61] closely.

We start by noting that subsumption is NP-complete [62]. This puts ILP in Σ_2^P-complete, the class of decision problems [why is this a decision problem now? isn't it a search problem?] that are in NP even if we have access to an oracle for NP [61]. The intuition behind this is simple: recall that in ILP we don't just try to subsume a single clause, but find an H subsuming all clauses in E^+ and none in E^-. We can do this with two oracles, one for guessing H and one for finding θ. The complexity result simply states that these two oracles are in NP and that they are necessary and sufficient.

Can we do any better by relaxing some of the assumptions? Yes, but not by much. Instead of aiming for exact, deterministic learnability, we can try to efficiently learn a hypothesis that is close enough to the true one with high probability. Making 'efficiently', 'close enough', and 'high probability' more precise leads to the definition of the class of problems that are "polynomially probably approximately correct learnable" or polynomially PAC-learnable. These are the problems for which there is an algorithm with the following properties:

- It runs in polynomial time in input size and inverse error.
- The coverage of the output hypothesis is highly likely to have substantial overlap with that of the true hypothesis.

This definition is not strictly about complexity since it also depends on the sampling distribution D. However, it serves as an intermediate step to establish whether a problem lies in RP, the class of decision problems for which a polynomial randomised algorithm with success probability $> 1/2$ exists if the answer is yes (see theorem 1 in [57]). The upshot is that polynomial PAC-learnability implies a problem is in RP [CITATION needed?].

- [61] contains a lot of negative results showing that many tweaks don't suffice to take ILP decision problems out of Σ_2^P

- [57] contains a single positive result (theorem 13): i j-determinate k-discriminative nonrecursive function-free predicate definitions are polynomially PAC-learnable under arbitrary distributions. This implies the problem is in RP (randomised polynomial).

4.2.4 Sample complexity

The strong inductive bias imposed by the background knowledge and the program template allow ILP systems to generalise well from small sample sizes, sometimes even managing one-shot learning [63].

The sample complexity of ILP in this setting has been rigorously studied by Cropper and Muggleton in the context of their proposed algorithm meta-interpretive learning (MIL). MIL is an approach to ILP that handles predicate invention through a series of metarules (second-order Horn clauses) [**?**]. Within this framework, the sample complexity of MIL for PAC-learning is linear in the hypothesis size and number of body literals contained in the metarules and logarithmic in the number of background predicates and the number of metarules (see Proposition 2 in [**?**]).

In deep learning, the relation between sample size and learning is more complex. Conventional wisdom in statistical learning theory roughly says that if a model has more parameters than it has training data, it will overfit [CITATION NEEDED]. However, empirical results in deep learning seem to contradict this, with deep neural networks having low test error even when trained on relatively small datasets. A recent attempt to reconcile these two phenomena invokes what has been called *double descent* [**?**]. According to this hypothesis, the test error is non-monotonic with respect to sample size in some settings: there is *critically parametrised regime* that happens when the number of training samples is approximately equal to some measure of model complexity. In this regime, more training or larger models hurt test error instead of improving it, as opposed to what happens in the under- and over-parametrised regimes. This phenomenon has been shown to hold across several popular deep learning models like CNNs or transformers.

4.2.5 Time complexity

Finding a hypothesis h with background B has complexity in $O\left((|h| \cdot |B|)^{|h|}\right)$ [3].

'function free universally quantified Horn expressions are exactly learnable... possible with membership and equivalence queries with resources polynomial in the number of clauses in the expression, though superpolynomial in the number of universally quantified variables. Similar results for related models including entailment queries and ILP are also derived.' [64]

Time or proof-depth bounding ...

4.2.6 Generalisation

One way of a machine learning system generalising well is: not overfitting, which means having a low test error relative to the training error.

A common way to avoid overfitting in ILP systems is to prevent too much specialisation (moving down the subsumption lattice toward concrete exemplars), in effect forcing the output hypothesis to become more abstract.

Overfitting is prevented in DL systems through a bag of tricks intended to overcome the 'laziness' of the training procedure (that is, the network's tendency to only learn what it necessary to represent a particular training set). Dropout, which randomly deletes connections between units, could be interpreted as abstracting over ... Another way to encourage abstraction in deep learning is regularization: adding layers to the network which penalize strong weight / (...)

A machine learning algorithm that can perform well with data that is taken out of a different distribution that its learning data is is said to perform well out of distribution. This is a form of generalization as well.

Some deep learning methods are set-up to perform well out of distribution [CITATION needed]. ILP is in general not able to handle data out of its training distribution. For example, if you train a system on traditional family data in which kids can have only one dad, then the output program may not be able to deal with data in which someone has two dads. (This is much worse in the case of mistaken background knowledge, for instance $\forall x, (\,\text{father}(x) \rightarrow \text{man}(x)\,)$ - since most systems take B to be absolutely certain.)

4.2.7 Transfer learning

Transfer learning deals with transferring knowledge gained in one learning task to another task. It deals with storing knowledge gained while solving one problem and applying it to a different but related problem.

In one sense, transfer learning is trivial for ILP. The output hypothesis is declarative and pure, and so it can be simply copied and pasted into the knowledge base of the next iteration of the problem, i.e., storing knowledge gained is trivial. Whether or not this learned hypothesis can be used in a different task depends on whether the tasks are different in a relevant way. Because of ILP's categorical nature and its sureness about the background, if there is a fact in the learned hypothesis that doesn't hold for the new task, then adding that hypothesis to the background of the new task is detrimental.

In DL one could take a model trained for one task, cut off its output layer and immediately apply it to other tasks. This can perform quite well [CITATION needed]. This process can also involve fine-tuning the weights on the new task. In other words, take the old network as initialization for the new task.

If I learned a lot about human eyes, can I translate this to knowing a lot about cat eyes? In DL, we could train a network to recognize human eyes, this network could be used as initialization for the neural network that should learn to recognize cat eyes. If the human eye hypothesis is too specific (contains things like "is not yellow", "has a round pupil", "is wider than it is high") then when used as B for the cat eye task, it would not be usable, and would merely slow the search (?).

5 Potential directions for ILP

In the previous sections we compared current ILP and DL. In this section we take a speculative view of the potential of ILP. To do so, we first look at the current limitations of ILP and then pinpoint potential improvements. Lastly, we look at what the combination of ILP and DL has to offer. We consider the safety properties and capabilities of hypothetical forms of ILP.

5.1 Current Limitations to ILP

1. *Naivety about noise and ambiguity.* In simple concept learners, a single mislabelled example can prevent learning entirely. However, progress has been made in handling noise and ambiguity: first, the low-hanging fruit of detecting mutual inconsistency in data (by deriving contradictions); and second limiting how far a top-down specialisation should go, when noise is assumed to be present. Further, the use of learned hypotheses for transfer learning between runs of ILP is limited by the noise a given learned program is likely to have picked up: current systems assume that background knowledge (like a transferred program) is certain.

2. *Unusually large resource requirements.* We discussed the terseness of ILP outputs as a virtue –but it is equally true that present systems cannot learn large theories given a practical level of resources. The space complexity of admissible search (that is, algorithms guaranteed to yield an optimum) is exponential in hypothesis length [27]. For this reason, predicate invention is limited even in state-of-the-art systems to, at most, dyadic predicates.

3. *Handcrafting of task-specific inductive biases.* Almost all ILP systems use some form of 'template' to generate the candidate clauses that form predicates (for instance 'mode declarations' in Progol and Aleph, meta-rules in Metagol, or rule templates in ∂ILP) [65]. In some sense these are hyperparameters, as found in most ML systems. But templates can be enormously informative, up to and including specifying which predicates to use in the head or body of h, the quantifier of each argument in the predicate, which arguments are to be treated inputs and outputs, and and so on. Often unavoidable for performance reasons, templates risk pruning unexpected solutions, involve a good deal of expert human labour, and lead to brittle systems which may not learn the problem structure, so much as they are passed it to begin with. This is an open problem, though recent work has looked at selecting or compressing a given set of templates [66], [24]. The ∂ILP authors also report an experiment with

generating templates, but the authors note that at least their brute-force approach is straightforwardly and permanently infeasible[10].

4. *Single answers with no traces.* Most ILP systems yield just one output program, and the heuristics used in high-performance systems must discard the counterfactual clauses (and paths of clauses), to constrain the search. However, nontrivial learning problems tend to have many aspects (i.e. ultimately unused clauses) of independent interest to the end-user. Consider the "top-5" outputs of modern computer vision classifiers, which give several calibrated answers for a given input. By default, ILP would give only the 'top-1' predicted label and offer only a sketch of the search process.

5. *Usability.* ILP systems remain a tool for specialists: to our knowledge, no user-friendly system has so far been developed.[5] To some extent this is due to the data representation: somewhat more than a working knowledge of first-order logic and (usually) Prolog are required to input one's own data. Whether this is a higher barrier than the basic linear algebra required for modern deep learning libraries, or merely a rarer skill in the data science community, would require empirical study. But the effect is the same: ILP seems more difficult to use.

6. *The deep threat to knowledge representation.* The knowledge representation programme faces a challenge from rapid progress in learned representations and end-to-end deep learning in computer vision, natural language processing, and other applied fields. Rich Sutton summarises this challenge as "the bitter lesson": that massively scaling dataset size and model size tends to outperform hand-crafting of features and heuristics by domain experts [67]. This is a 'limitation' of ILP insofar as it cannot itself follow suit and take advantage of learned representations to the same degree.

5.2 Potential Improvements to ILP

As discussed in Section 5.1, there are shortcomings in common to all extant ILP systems. We list aspects of ILP that could mitigate these:

1. Shrinking the target theory
2. Reducing the background size
3. Data compression: Unification of examples or literals
4. New search algorithms
5. Handling noise and ambiguity (Response noise or attribute noise)

[5] The RDM Python wrapper is a partial exception, `https://github.com/xflows/rdm`

The first three approaches all relate to the fact that ILP takes longer to run if it has more data to parse. The order of finding a hypothesis h with background B is in $O\left((|h| \cdot |B|)^{|h|}\right)$. Thus if we shrink the target theory $|h|$ or reduce the background $|B|$, we drastically speed up the algorithm. In general, data compression leads to ...

Reducing the target theory is an active line of research [cite cropper and more ?]. In [**?**], the target theory is reduced through ... Other ways in which it could be reduced include ...

Examples of ways in which the background could be reduced.

Just as an exponential increase in computation power enabled the boom in large neural networks, so too might inductive proof search benefit from similar algorithms to exploit this increase. Another route is to import general improvements from other areas of ML, for instance: parallelism in computation [68] and modelling [69], population-based training [70], ensembles of weak learners [37], curriculum learning [71],

5.3 Neural-Symbolic Systems: Integrated or Modular?

In the wider field of AI, an extraordinary amount of recent work has tried to unify connectionist and symbolic methods, mostly in the form of trying to implement relational or logical reasoning on neural networks, chasing breakthrough capabilities a la ImageNet [TODO CITATION] or GPT-2 [TODO CITATION]. From the safety perspective, this unification often loses the properties we originally targeted.

When looking at the safety properties of neural-ILP systems we distinguish between integrated and modularised systems. By integrated we mean an ILP algorithm that uses subsymbolic processing, the whole way from input until output. By modularised we mean a system that has pure ILP modules alternated with neural modules. e.g. preprocessing.

In Section 5.2, we discussed potential improvements that can be made to ILP to make it more competitive with other ML methods. Some of these improvements require only attaching a preprocessing module to the main ILP module, which likely won't have much impact on the interpretability of the algorithm as a whole. One way to view this is that preprocessing changes the problem that needs to be solved, but doesn't change the algorithm that is used to solve it. Other forms of modular neural-ILP systems include student-teacher distillation, where we have a core ILP student module and a neural teacher module [72].

On the other hand, end-to-end differentiable versions of ILP are fundamentally different algorithms in that they are no longer based on a purely logical non-probabilistic core ILP module. A differentiable version of ILP is introduced in

[10]. This approach and similar integrated neural-symbolic versions of ILP may lose some of the interpretability properties, which we think are its main safety advantage, but an integrated neural approach may be necessary to deal with ambiguity.

6 Conclusion

If the sample complexity of deep learning continues to fall [CITATION NEEDED], or if learning deep representations for structured or symbolic data continues to make ground [CITATION NEEDED], then ILP will have lost its comparative advantage, since efficiency and knowledge representation are its two most promising capabilities.

7 Appendix A: Other dimensions of ILP systems

Table 1 lists those properties that are most informative for understanding how recent ILP systems differ. Others include:

1. *Predictive vs Descriptive domain.* all of the above are predictive systems, i.e. they aim at generalisation and produce reusable models.
2. *Completeness*: Is the search guaranteed to find a correct hypothesis, if one exists? (For instance, Progol is provably not complete. [CITATION])
3. *Incremental* (or online learning): can new examples and background be added during learning?
4. *Interactive* (or active learning): can it query an oracle for guided search? Interactive entails incremental.
5. *The data 'setting'*: whether examples are constrained to contain only literals ("the example setting") or can contain predicates (e.g. "the non-monotonic setting").
6. *Single vs multiple concept-learning*: Early systems were limited to 'single predicate learning', in which all examples are taken as evidence for one, singleton predicate; from FOIL onwards we have multiple predicate learning, where several concepts are learned with their relations from one dataset.
7. *Numerical data*: Early systems could not make use of numerical data, or handle intervals.
8. *Semantics*: all of the above use the 'definite' semantics, limiting the number of positive literals in the head to 1, and so avoiding the need for negation-as-failure.
9. *Clause-level or theory-level search*: whether the search proceeds at the clause-level (see Algorithms 2 and 3) or theory-level (see Algorithm 1).
10. *Robustly learning recursion*: the degree of recursion that can be learned; approaches which rely on bottom clause construction [27] cannot learn recursive clauses without examples of both the base case and the inductive case, in that order.
11. *Problem domain*: Some approaches first transform the ILP problem to the propositional[73] or SAT domain [74], rather than solving the original relational search problem. It's unclear if this is still an ILP system afterward, but it is at very least a close neighbour.
12. Another degree of freedom is whether negative examples are specified at all, or merely inferred as the complement of the minimal Herbrand model.

8 Appendix B: key ILP algorithms

Generic algorithm: ILP as two-part search

```
**AbstractILP** (allowed inference rules $R$, language bias $\mathcal{L}$) :
**Result:** hypothesis $h$, a rule in first-order predicate logic

$H \leftarrow$ initialHypotheses($\mathcal{L}$) ;
while not stop-criterion do
    $h \leftarrow$ Pop($H$);
    $r_1, \cdots, r_k \leftarrow$ ChooseRules($R$) ;
    $h_1, \cdots, h_n \leftarrow$ Apply($r_1, \cdots, r_k$, $h$) ;
    $H \leftarrow H + \{h_1, \cdots, h_n\}$;
    $H \leftarrow$ Prune($H$);
end
return $h \leftarrow$ Pop($H$);
```

Algorithm 1: adapted from Muggleton & de Raedt (1994) [13]

At this level of abstraction (which overlooks the data, even), we see the structure and parameters of ILP in general:

- `initialHypotheses`: defines the hypothesis space. High-performance ILP systems generally begin with only one hypothesis in this queue ($B \wedge E^+$).
- `stop-criterion`. Terminating condition. Examples include: finding a strict solution; meeting some threshold on the statistical significance of the heuristic score of a hypothesis.
- `Pop`: which hypothesis to try next. This is half of the search procedure. It can proceed by simple classical search, e.g. LIFO (breadth-first) or FIFO (depth-first) or priority queue (best-first), or by heuristics (see `Prune`).
- `ChooseRules` and `Apply`: determines which inference rules to use on H, for instance absorption, addClause, dropNegativeLiteral. These are generally syntactic modifications of h, and yields a set of derived hypotheses $\{h_1 \cdots h_n\}$.
- `Prune`: which candidate hypotheses to delete from H; the other half of the search. We score the hypotheses, for instance using estimated probabilities $p(H|B \wedge E)$. (Note that, in an exact system, generalisation and specialisation allow us to simultaneously assign $p = 0$ to all specialisations of hypotheses that fail to entail a positive, and $p = 0$ to all generalisations of hypotheses $\wedge$ example e^- that entail the empty clause.) An alternative score is the minimum compression length of h.

Top-down search: The First-Order Inductive Learner

```
FOIL (background knowledge $B$,
      positive examples $E^+$,
      negative examples $E^-$) :
Result: hypothesis $h$, a rule in first-order predicate logic

$h \leftarrow \emptyset$
while $E^+$ not empty do
    clause $\leftarrow$ LearnNewClause($E^+, E^-, h$)
    candidateTheory $\leftarrow B \wedge h \wedge$ clause
    coveredPositives $\leftarrow \{e \in E^+ :$ candidateTheory $\vdash e\}$
    $E^+ \leftarrow E^+ \setminus$ coveredPositives
    $h \leftarrow h \cup \{$clause$\}$
end
return $h$
```

```
LearnNewClause($E^+, E^-, h$) :

clause $\leftarrow \emptyset$
while $E^-$ not empty do
    bestLiteral $\leftarrow \arg\max_l$ Gain(clause, $l, E^+, E^-$)
    clause $\leftarrow$ clause $\cup \{$bestLiteral$\}$
    candidateTheory $\leftarrow B \wedge h \wedge$ clause
    coveredNegatives $\leftarrow \{e \in E^- :$ candidateTheory $\vdash e\}$
    $E^- \leftarrow E^- \setminus$ coveredNegatives
end
return clause
```

```
Gain(clause, literal, $E^+, E^-$) :

posCoveredBefore $\leftarrow$ Satisfies(clause, $E^+$)
posCoveredAfter $\leftarrow$ Satisfies(clause $\wedge$ literal, $E^+$)
negCoveredBefore $\leftarrow$ Satisfies(clause, $E^-$)
negCoveredAfter $\leftarrow$ Satisfies(clause $\wedge$ literal, $E^-$)

posPreserved $\leftarrow$ posCoveredBefore $\cap$ posCoveredAfter
gainAfter $\leftarrow \log_2(\frac{\#\text{posCoveredAfter}}{\#\text{posCoveredAfter}+\#\text{negCoveredAfter}})$
gainBefore $\leftarrow \log_2(\frac{\#\text{posCoveredBefore}}{\#\text{posCoveredBefore}+\#\text{negCoveredBefore}})$

return #posPreserved $\times$(gainAfter $-$ gainBefore)
```

Algorithm 2: adaptation of FOIL [26][75] with gain heuristic

Bottom-up search: The default Aleph algorithm

```
**Aleph**(background knowledge $B$,
          positive & negative examples $E$,
          mode declaration $\mathcal{L}$) :
Result: hypothesis $h$

$h \leftarrow$ empty clause ;
while $E$ is not empty do
    $e \leftarrow$ Select($E$) ;
    $\perp_e \leftarrow$ BottomClause($e, \mathcal{L}$) ;
    $c^* \leftarrow$ ClauseReduction($\perp_e$) ;
    $h \leftarrow h + c^*$ ;
    $E \leftarrow$ Prune($E$);
end
return $h$
```

```
ClauseReduction(most specific clause $\perp$) :
Result: a clause $c$ more general than $\perp$

activeSet $\leftarrow \{\}$
bestSolutionCost $\leftarrow \infty$
currentBestClause $\leftarrow \perp$

while activeSet is not empty do
    clause $\leftarrow$ Pop(activeSet)
    children = $\{i = 1, ..., n_k$ : child$_i\} \leftarrow$ GenerateChildren(clause)
    $C_i \leftarrow$ Cost(children)
    $L_i \leftarrow$ LowerBoundCost(children)

    for $i = 1, ..., n_k$ do
        if $L_i \geq$ bestSolutionCost then
            children $\leftarrow$ children $-$ child$_i$
        end
        else
            if child$_i$ is a complete solution and $C_i <$ bestSolutionCost
             then
                bestSolutionCost $\leftarrow C_i$
                currentBestClause $\leftarrow$ child$_i$
                activeSet $\leftarrow \{i \in$ activeSet : $L_i \leq C_i\}$
            end
            activeSet $\leftarrow$ activeSet + child$_i$
        end
    end
end
return currentBestClause
```

Algorithm 3: Adapted from Srinivasan (1999) [28].

- `Select`: pick an example e to be generalised.
- `BottomClause`: 'saturate' the example - find the most specific clause that both entails e and obeys the mode declaration (that is, the specified language bias). The procedure is from [27].
- `ClauseReduction`: search for a subset (or derived subset) of the literals in $\perp_e$ with the best score, $\arg\max_c$ Score($\perp_e$). This is a branch-and-bound algorithm, and returns a search tree with clauses for nodes. As standard, this is the gain in accuracy of adding c to h. A 'good' clause is defined by hyperparameters: a maximum clause length, a minimum accuracy, a minimum 'child weight' (i.e. a threshold number of positive examples covered by c before c can be selected).
- `Prune`: remove examples (and background) made redundant under the new hypothesis h. Unlike Algorithm 1, this prunes the *examples* rather than the candidate hypotheses.

References

[1] Leilani Gilpin; David Bau; Ben Yuan; Ayesha Bajwa; Michael Specter; Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. *IEEE 5th International Conference on Data Science and Advanced Analytics*, 2018. `https://arxiv.org/pdf/1806.00069.pdf`.

[2] Gary Marcus. Deep learning: A critical appraisal. *arXiv preprint*, 2018. `https://arxiv.org/abs/1801.00631`.

[3] Jose Hernandez-Orallo. Deep knowledge: Inductive programming as an answer. 2018. `https://dmip.webs.upv.es/papers/deepknowledge2013.pdf`.

[4] Michael Siebers. inductive-programming community site. `https://inductive-programming.org/people.html`, 2019.

[5] BAIR. Berkeley AI Research lab, list of students. `https://bair.berkeley.edu/students.html`, 2020.

[6] Claude Sammut. The origins of inductive logic programming: A prehistoric tale. *Proceedings of the 3rd international workshop on inductive logic programming*, 1993. `http://www.doc.ic.ac.uk/~shm/Papers/sammut_mlj.pdf`.

[7] Stephen Muggleton; Luc De Raedt; David Poole; Ivan Bratko; Peter Flach; Katsumi Inoue; Ashwin Srinivasan. ILP turns 20: Biography and future challenges. *Machine Learning*, 2011. `https://link.springer.com/content/pdf/10.1007%2Fs10994-011-5259-2.pdf`.

[8] Andrew Cropper; Rolf Morel; Stephen Muggleton. Learning higher-order logic programs. *Machine Learning*, 2019. `https://link.springer.com/article/10.1007/s10994-019-05862-7`.

[9] Luc De Raedt; Kristian Kersting. Probabilistic inductive logic programming. *Probabilistic Inductive Logic Programming*, page 1–27, 2008.

[10] Richard Evans; Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, pages 1–64, 2018. `https://www.jair.org/index.php/jair/article/view/11172/26376`.

[11] James Hawthorne. Inductive logic, stanford encyclopaedia of philosophy. `https://plato.stanford.edu/entries/logic-inductive/`, 2018.

[12] Peter Flach. Simply logical: Intelligent reasoning by example. *Wiley*, 1994. `https://book.simply-logical.space/`.

[13] Stephen Muggleton; Luc de Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, pages 629–679, 1994. `https://www.sciencedirect.com/science/article/pii/0743106694900353`.

[14] Wim van Laer; Luc de Raedt. How to upgrade propositional learners to first order logic: A case study. *Relational Data Mining*, 2001. `https://link.springer.com/chapter/10.1007/978-3-662-04599-2_10`.

[15] Peter Clark; Tim Niblett. The cn2 induction algorithm. *Machine Learning*, pages 261–283, 1989. `https://link.springer.com/article/10.1023/A:1022641700528`.

[16] Stefan Kramer; Gerhard Widmer. Inducing classification and regression trees in first order logic. *Relational Data Mining*, pages 140–159, 2001. `https://link.springer.com/chapter/10.1007/978-3-662-04599-2_6`.

[17] Jan Ramon; Maurice Bruynooghe. A framework for defining distances between first-order logic objects. *Proceedings of the eighth international conference on inductive logic programming*, 1998. `https://link.springer.com/chapter/10.1007/BFb0027331`.

[18] Reza Farid; Claude Sammut. Plane-based object categorisation using relational learning. *Machine Learning*, 2013. `https://link.springer.com/article/10.1007/s10994-013-5352-9`.

[19] Frank DiMaio; Jude Shavlik. Learning an approximation to inductive logic programming clause evaluation. *Proceedings of the 14th International Conference on Inductive Logic Programming*, 2004. `http://ftp.cs.wisc.edu/machine-learning/shavlik-group/dimaio.ilp04.pdf`.

[20] Thomas Elsken; Jan Hendrik Metzen; Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 2018. http://jmlr.org/papers/v20/18-598.html.

[21] Ian Witten; Eibe Frank; Mark Hall; Christopher Pal. Data mining: Practical machine learning tools and techniques. *Morgan Kaufmann*, 2017. https://www.sciencedirect.com/book/9780128042915/data-mining.

[22] Guillermo Valle-Pérez; Chico Camargo; Ard Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. *arXiv preprint*, 2018. https://arxiv.org/abs/1805.08522.

[23] Kenji Kawaguchi; Leslie Pack Kaelbling; Yoshua Bengio. Generalization in deep learning. *arXiv preprint*, 2017. .

[24] Andrew Cropper; Sophie Tourret. Logical reduction of metarules. *Machine Learning*, 2019. https://link.springer.com/article/10.1007/s10994-019-05834-x.

[25] Sašo Džeroski. List of ILP systems. http://www-ai.ijs.si/~ilpnet2/systems/, 2002.

[26] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 1990. https://link.springer.com/article/10.1023/A:1022699322624.

[27] Stephen Muggleton. Inverse entailment and progol. *New Generation Computing*, 1995.

[28] Ashwin Srinivasan. The Aleph system. https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html, 1999-2006.

[29] Andrew Cropper; Stephen Muggleton. The Metagol system. https://github.com/metagol/metagol, 2016.

[30] Mark Law; Alessandra Russo; Krysia Broda. Inductive learning of answer set programs. *European Workshop on Logics in Artificial Intelligence*, 2014. https://link.springer.com/chapter/10.1007/978-3-319-11558-0_22.

[31] Stephen Muggleton; Ute Schmid; Christina Zeller; Alireza Tamaddoni-Nezhad; Tarek Besold. Ultra-strong machine learning: comprehensibility of programs learned with ILP. *Machine Learning*, pages 1119–1140, 2018. https://link.springer.com/article/10.1007/s10994-018-5707-3.

[32] Chris Olah; Nick Cammarata; Ludwig Schubert; Gabriel Goh; Michael Petrov; Shan Carter. Zoom in: An introduction to circuits. https://distill.pub/2020/circuits/zoom-in/, 2020.

[33] Evan Hubinger. Chris Olah's views on agi safety. https://www.alignmentforum.org/posts/X2i9dQQK3gETCyqh2/chris-olah-s-views-on-agi-safety, 2019.

[34] Elaine Angelino; Nicholas Larus-Stone; Daniel Alabi; Margo Seltzer; Cynthia Rudin. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 2018. https://arxiv.org/pdf/1704.01701.pdf.

[35] Benjamin Letham; Cynthia Rudin; Tyler McCormick; David Madigan. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 2015. https://projecteuclid.org/download/pdfview_1/euclid.aoas/1446488742.

[36] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 2019. https://www.nature.com/articles/s42256-019-0048-x.

[37] Ning Jiang; Simon Colton. Boosting descriptive ILP for predictive learning in bioinformatics. *Inductive Logic Programming, ILP 2006*, 2007. https://link.springer.com/chapter/10.1007/978-3-540-73847-3_28.

[38] Guy Katz; Clark Barrett; David Dill; Kyle Julian; Mykel Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *Computer Aided Verification. Lecture Notes in Computer Science, vol 10426. Springer, Cham*, 2017. .

[39] Shiqi Wang; Kexin Pei; Justin Whitehouse; Junfeng Yang; Suman Jana. Efficient formal safety analysis of neural networks. *Advances in Neural Information Processing Systems 31*, 2018. .

[40] Gagandeep Singh; Timon Gehr; Markus Püschel; Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.3, POPL, Article 41*, 2019. .

[41] Elena Botoeva; Panagiotis Kouvaros; Jan Kronqvist; Alessio Lomuscio; Ruth Misener. Efficient verification of relu-based neural networks via dependency analysis. *AAAI*, 2020. .

[42] Kai Y. Xiao; Vincent Tjeng; Nur Muhammad Shafiullah; Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. *International Conference on Learning Representations (ICLR)*, 2019. .

[43] RD King; S Muggleton; RA Lewis; MJ Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the national academy of sciences*, 1992. .

[44] A Srinivasan; S Muggleton; RD King; MJE Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. *Proceedings of the 4th international workshop on inductive logic programming*, 1994. .

[45] RD King; A Srinivasan; MJE Sternberg. Relating chemical activity to structure: An examination of ILP successes. *New Generation Computing*, 1996. .

[46] A Srinivasan; RD King; SH Muggleton; MJE Sternberg. Carcinogenesis predictions using ILP. *International Conference on Inductive Logic Programming*, 1997. .

[47] R; Kakas A; Muggleton S Tamaddoni-Nezhad, A; Chaleil. Application of abductive ILP to learning metabolic network inhibition from temporal data. *Mach. Learn.*, 2006. .

[48] Kazuhisa Tsunoyama; Ata Amini; Michael Sternberg; Stephen Muggleton. Scaffold hopping in drug discovery using inductive logic programming. *Journal of Chemical Information and Modeling*, 2008. .

[49] Hoan Nguyen; Tien-Dao Luu; Olivier Poch; Julie D. Thompson. Knowledge discovery in variant databases using inductive logic programming. *Bioinformatics and Biology Insights*, 2013. .

[50] R. Kaalia; A. Srinivasan; A. Kumar; I. Ghosh. ILP-assisted de novo drug design. *Maching Learning*, 2016. `https://link.springer.com/article/10.1007/s10994-016-5556-x`.

[51] S.H.; Sternberg M.J.E. Turcotte, M.; Muggleton. Automated discovery of structural signatures of protein fold and function. *Mol. Biol.*, 2001. .

[52] S.H.; Sternberg M.J.E. Turcotte, M.; Muggleton. The effect of relational background knowledge on learning of protein three-dimensional fold signature. *Machine Learning*, 2001. .

[53] S.H.; Sternberg M.J.E. Cootes, A.P.; Muggleton. The automatic discovery of structural principles describing protein fold space. *Mol. Biol.*, 2003. .

[54] Andrew W Senior; Richard Evans; John Jumper; James Kirkpatrick; Laurent Sifre; Tim Green; Chongli Qin; Augustin Žídek; Alexander W R Nelson; Alex Bridgland; Hugo Penedones; Stig Petersen; Karen Simonyan; Steve Crossan; Pushmeet Kohli; David T Jones; David Silver; Koray Kavukcuoglu; Demis Hassabis. Improved protein structure prediction using potentials from deep learning. *Nature*, 2019. .

[55] S. Tärnlund. Horn clause computability. *BIT Numerical Mathematics*, 1977. `https://doi.org/10.1007/BF01932293`.

[56] Jerzy Marcinkowski; Leszek Pacholski. Undecidability of the horn-clause implication problem. *IEEE Symposium on Foundations of Computer Science 1992*, 1992. `https://open.bu.edu/bitstream/handle/2144/1457/1993-004-unif93proc.pdf?sequence=1#page=86`.

[57] Sašo Kietz, Jörg-Uwe; Džeroski. Inductive logic programming and learnability. *ACM SIGART Bulletin*, 5(1):22–32, 1994.

[58] David Silver; Julian Schrittwieser; Karen Simonyan; Ioannis Antonoglou; Aja Huang; Arthur Guez; Thomas Hubert; Lucas Baker; Matthew Lai; Adrian Bolton; Yutian Chen; Timothy Lillicrap; Fan Hui; Laurent Sifre; George van den Driessche; Thore Graepel; Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 2017. `https://www.nature.com/articles/nature24270.epdf`.

[59] Madhu Advani; Andrew Saxe. High-dimensional dynamics of generalization error in neural networks. *arXiv preprint*, 2017. `https://arxiv.org/pdf/1710.03667.pdf`.

[60] Russell Impagliazzo. A personal view of average-case complexity. *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, 1995. `https://ieeexplore.ieee.org/document/514853`.

[61] Georg Gottlob, Nicola Leone, and Francesco Scarcello. On the complexity of some inductive logic programming problems. In *International Conference on Inductive Logic Programming*, pages 17–32. Springer, 1997.

[62] Shan-Hwei Nienhuys-Cheng; Ronald de Wolf. Foundations of inductive logic programming. 1997.

[63] Dianhuan Lin; Eyal Dechter; Kevin Ellis; Joshua Tenenbaum; Stephen Muggleton. Bias reformulation for one-shot function induction. *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, 2014. `https://dl.acm.org/doi/10.5555/3006652.3006741`.

[64] Roni Khardon. Learning first order universal horn expressions. *COLT 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 154–165, 1998. `https://dl.acm.org/doi/10.1145/279943.279976`.

[65] Ali Payani; Faramarz Fekri. Inductive logic programming via differentiable deep neural logic networks. *CoRR*, abs/1906.03523, 2019.

[66] Andrew Cropper; Stephen Muggleton. Logical minimisation of meta-rules within meta-interpretive learning. *Inductive Logic Programming 24th International Conference*, 2014. `https://link.springer.com/chapter/10.1007/978-3-319-23708-4_5`.

[67] Richard Sutton. The bitter lesson. `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`, 2019.

[68] David Page; Ashwin Srinivasan. ILP: A short look back and a longer look forward. *Journal of Machine Learning Research*, page 415–430, 2003. https://dl.acm.org/doi/10.5555/945365.945390.

[69] Samyam Rajbhandari; Jeff Rasley; Olatunji Ruwase; Yuxiong He. ZeRo: Memory optimization towards training a trillion parameter models. https://arxiv.org/abs/1910.02054, 2020.

[70] Max Jaderberg; Valentin Dalibard; Simon Osindero; Wojciech M. Czarnecki; Jeff Donahue; Ali Razavi; Oriol Vinyals; Tim Green; Iain Dunning; Karen Simonyan; Chrisantha Fernando; Koray Kavukcuoglu. Population based training of neural networks. *arXiv preprint*, 2017. https://arxiv.org/abs/1711.09846.

[71] Hank Conn; Stephen Muggleton. The effect of predicate order on curriculum learning in ILP. *Late Breaking Papers of ILP 2017*, 2017. http://ceur-ws.org/Vol-2085/connLBP-ILP2017.pdf.

[72] Zhiting Hu; Xuezhe Ma; Zhengzhong Liu; Eduard Hovy; Eric Xing. Harnessing deep neural networks with logic rules. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016. https://www.aclweb.org/anthology/P16-1228.pdf.

[73] Stefan Kramer; Nada Lavrač; Peter Flach. Propositionalization approaches to relational data mining. *Relational Data Mining*, 2001. https://link.springer.com/chapter/10.1007/978-3-662-04599-2_11.

[74] Noriaki Chikara; Miyuki Koshimura; Hiroshi Fujita; Ryuzo Hasegawa. Inductive logic programming using a maxsat solver. *25th International Conference on Inductive Logic Programming*, 2015. http://www.ilp2015.jp/papers/ILP2015_submission_35.pdf.

[75] Vinay K. Chaudhri. Inductive logic programming. http://web.stanford.edu/~vinayc/logicprogramming/html/inductive_logic_programming.html, 2019.