Safety Properties of Inductive Logic Programming

Gavin Leech, [*1] **Nandi Schoots,** [*2] **Joar Skalse** [3]
[1] University of Bristol
[2] King's College London and Imperial College London
[3] University of Oxford
[*] *Equal contribution*
g.leech@bristol.ac.uk

Abstract

This paper investigates the safety properties of inductive logic programming (ILP), particularly as compared to deep learning systems. We consider the following properties: ease of model specification; robustness to input change; control over inductive bias; verification of specifications; post-hoc model editing; and interpretability. We find that ILP could satisfy many of these properties in its home domains. Lastly, we propose a hybrid system using ILP as a preprocessor to generate specifications for other ML systems.

Introduction

Symbolic approaches to AI are sometimes considered safer than neural approaches (Condry 2016; Anderson et al. 2020). We investigate this by analysing how one symbolic approach, inductive logic programming (ILP), fares on specific safety properties.

ILP is a declarative subfield of ML for learning from examples and encoded "background knowledge" (predicates and constraints), using logic programs to represent both these inputs and the output model (Muggleton 1999). (We use 'output model' and 'ILP hypothesis' interchangeably.)

We find ILP to have potential to satisfy an array of safety properties. To arrive at this, we survey existing work in ILP and deep learning in light of the safety properties defined in the framework of Ortega and Maini (2018). We also formalise *robustness to input change* and *model editing*. We suggest a hybrid system, in which ILP is used as a preprocessing step to generate specifications for other ML systems.

To our knowledge, this is the first analysis of ILP's safety potential, and of ILP's differences from deep learning. Related work includes Cropper, Dumančić, and Muggleton (2020)'s recent survey of ILP, the interpretability work of Muggleton et al. (2018b), and Powell and Thévenod-Fosse (2002)'s study of rule-based safety-critical systems.

Consider a machine learning system 'safe' when the system's goals are specified correctly, when it acts robustly according to those goals, when we are assured about these two properties (Ortega and Maini 2018), such that the risk of harm from deploying the system is greatly reduced. ILP may be a natural fit for the *assurance* side of safety: often, not just the output model, but also the learning process takes place at a relatively high level (that is, at the level of symbolic inference). Similarly, ILP plausibly satisfies multiple important *specification* and *robustness* properties. We assess ILP on: **Specification** properties (ease of model specification and value loading; ease of adjusting the learned model to satisfy specifications; and control over inductive bias); **Robustness** properties (robustness to input change and to post-training model edits); **Assurance** properties (interpretability and explainability; verification of specifications; and control over the inductive bias).

Many safety properties await formalisation, preventing quantitative comparisons. Where a formal metric is lacking, we qualitatively compare ILP to deep learning (DL).

In the following we refer to 'ILP' as if it was monolithic, but ILP systems differ widely in search strategy, exactness, completeness, target logic (e.g. Prolog, Datalog, ASP), noise-handling, ability to invent predicates, and the order of the output theory (Boytcheva 2002). This diversity limits the general statements we can make, but some remain.

Safety properties of ILP

Model Specification

The specification of an ML system serves to define its purpose. When this purpose is successfully implemented in hard constraints, we may obtain guarantees about the system's behaviour. A defining feature of ILP systems is user-specified background knowledge. This provides a natural way to impose specifications on ILP systems. An ILP problem specification is a set of positive examples, negative examples, and background knowledge.

Consider two important properties of classical ILP. Given a background B, an output model M, and positive examples E^+, the model M is

- *weakly consistent* if: $B \wedge M \not\models \text{False}$; and
- *strongly consistent* if: $B \wedge M \wedge E^+ \not\models \text{False}$.

Weak consistency forbids the generation of models that contradict any clause in B (Muggleton 1999). In general, ILP algorithms must satisfy weak consistency (Muggleton 1999), though probabilistic systems allow its violation; see below. Hence, to guarantee that the learned model M satisfies some specification s, all we need to do is encode s in first-order logic (FOL) and add it to the background B. However, there are still some specification challenges for ILP.

Not all systems respect strong consistency. Many modern implementations of ILP are designed to handle noise in the example set (Srinivasan 2006; Muggleton et al. 2018a). For specifications encoded in the example set, noise handling means that the system is only nudged in the direction of the specification. Furthermore, probabilistic ILP systems can specify the background as probabilistic facts (De Raedt et al. 2015). This means that even weak consistency can be violated. As such, these systems may not offer specification guarantees.

Incompleteness. Even though a model satisfying our specification exists, an incomplete ILP algorithm might not find it. Some leading implementations of ILP are incomplete, i.e. a solution may exist even though the system does not find one (Cropper and Tourret 2018)

Specifications may be hard to encode as FOL formulae. In computer vision, a long tradition of manually encoding visual concepts was rapidly outperformed by learned representations (Goodfellow, Bengio, and Courville 2016): it proved possible to learn these improved concepts, but intractable to hand-code them. Insofar as ILP backgrounds must at present be manually encoded (as opposed to learned via predicate invention), we infer that some specifications are not practically possible to impose on ILP.

Human values are hard to encode as FOL formulae. A particularly interesting kind of specification are those that concern norms or values, i.e. specifications that aim to ensure that the output respects ethical considerations. There is precedent for formalizing norms and moral obligations using logic – *deontic logic* is an area of philosophical logic that aims to formalise and deduce moral claims (McNamara 2019). This has been used to partially formalise some ethical frameworks (Kroy 1976; Peterson 2014). However, encoding general normative requirements in formal logic is an open problem. Further, we do not have a complete articulation of all such requirements in any formalism. It seems unlikely that in the near future we will obtain a complete encoding, owing to deep inconsistencies across people and the contextual nature of value (Yudkowsky 2011). Furthermore, it may be impossible to learn a representation of these preferences, in the absence of a strong model of human error (Armstrong and Mindermann 2018).

Model specification in DL. Methods exist for limited model specification in DL (Platt and Barr 1988), many of which focus on specific domains (Kashinath, Marcus et al. 2019; Zhang et al. 2020). However, if we interpret a specification as a hard constraint on outputs, then most current DL methods do not allow specification. Instead they impose soft constraints, modifying the loss to discourage out-of-specification behaviour. Imposing hard constraints in DL amounts to imposing a linear set of constraints on the output of the model. Soft constraints in the form of subtle alterations to the loss function or learning algorithm are harder to specify than e.g. a linear set of hard constraints (Pathak, Krähenbühl, and Darrell 2015). Soft constraints are pervasive due to the computational expense of hard constraints in neural networks: since networks can have millions of adaptive parameters, it is not practical to use ordinary constrained optimisation methods to impose them (Márquez-Neila, Salzmann, and Fua 2017).

Robustness to Input Change

Robustness concerns smooth output change: If we change the input slightly, will the output (of the learning algorithm or of the learned model) change only slightly? To formalize this, we define similarity of inputs and output hypotheses.

In DL input datasets, the problem description is usually very correlated with the semantics of the problem. For example, Gaussian noise usually does not affect the semantics of the problem. DL models are often insensitive to small changes in the description of the input. However, adversarial changes induce large changes in output, despite the input changes being trivial to the human eye (Szegedy et al. 2014).

For Horn clauses (a typical form in ILP output hypotheses), one distance measure is the 'rewrite distance' (the minimum number of syntactic edits that transform one clause into another) (Edelmann and Kunčak 2019). For our purposes, this is inappropriate, since it neglects the semantic distances we are targeting: a negation of the whole clause would count as a rewrite distance of 1, despite being maximally semantically different.

Definition 1 (Similarity of Datasets) *Given two datasets D_1, D_2, let H_1 and H_2 be the sets of hypotheses compatible with D_1 and D_2 respectively. Let the weight of a set of hypotheses H be defined as a weighed sum of the hypotheses in H, where more complex hypotheses are given lower weight (so that hypothesis h has weight $0.5^{2c(h)}$, where $c(h)$ is the complexity of h). We then say that D_1 and D_2 are similar if $H_1 \cap H_2$ has a large weight.*

Definition 2 (Similarity of Hypotheses) *We say that two hypotheses h_1 and h_2 are similar if the probability that they will agree on an instance x sampled from the underlying data distribution is high.*

Definition 3 (Robustness to Input Change) *Let $\mathcal{L}\colon \mathcal{D} \to \mathcal{M}$ be a learning algorithm. We say that $\mathcal{L}$ is robust to input change if it is the case that $\mathcal{L}(D_1)$ and $\mathcal{L}(D_2)$ are similar whenever D_1 and D_2 are similar. More specifically, we say that $\mathcal{L}$ has robustness parameters r_D, r_M if: for any D_1 and D_2 such that they have similarity r_D or higher, the similarity between $\mathcal{L}(D_1)$ and $\mathcal{L}(D_2)$ is at least r_M.*

We note that, for this notion of similarity between datasets, the distance between two ILP problems may be very large even if their descriptions are almost the same. For example, adding a negation somewhere in the description of D_1 may completely change its distance to D_2.

ILP is robust to syntactic input change. ILP is largely invariant to how the input problem is represented (in the sense of symbol renaming or syntactic substitutions, which do not affect the semantics). Two semantically equivalent problems have identical sets of compatible output hypotheses.

Examples of trivial syntactic changes to a problem include: renaming atoms or predicates; substituting a ground term for a variable; or substituting in a different variable. An

ILP problem statement is parsed as an ordered set of logical sentences, and substitutions within these sentences do not affect the semantics of the individual examples. Absent complicating implementation details, they thus do not affect the semantics of the output. Another syntactic change to a problem is adding or removing copies of examples; these changes do not have any effect on what hypothesis is output.

Changing the order of examples could (depending on the search algorithm) change the chosen output hypothesis. Even though the set of consistent output hypotheses does not change when the order of examples changes, the hypothesis that comes up first in the search may change. For example, Metagol depends on the order (Cropper and Morel 2020). This order dependence is a property of some clause-level search methods (Srinivasan 2006).

Robustness to semantic input change. Naturally, semantic changes to the problem can completely change the output hypothesis. For example, negating a single example can preclude finding any appropriate hypothesis.

Suppose D_1 and D_2 are two datasets, with corresponding hypothesis spaces respectively H_1 and H_2. ILP has a fixed order (which depends on the inductive biases) of traversing the total set of potential hypotheses for a solution. Say ILP outputs hypothesis h_1 for problem D_1 and hypothesis h_2 for D_2. Even if $H_1 \neq H_2$, h_1 and h_2 may be the same. When $h_1 \neq h_2$, we would like to assess their similarity.

Given an output model. If we change one input example, then we may be able to check whether this input example is consistent with the output model. We may not be able to completely visualise the coverage, but may be able to predict whether the output model will be different.

Empirically assessing robustness to input change. Potentially, sampling can inform us about the robustness to input change of ILP and deep learning. An experiment could work as follows: Generate ILP problems such that we (approximately) know the distance between the datasets. Then run ILP on each problem and store their output hypotheses. We then select a distance measure and assess the distance between each of the output hypotheses. This allows us to (approximately) evaluate the robustness to input change of ILP. A similar sampling process can be used for other learning algorithms to compare the robustness of different algorithms.

Control over Inductive Bias

The inductive bias of a learning algorithm is the set of (often implicit) assumptions used to generalise a finite input set to a complete output model (Mitchell 1980; Hüllermeier, Fober, and Mernberger 2013). If several hypotheses fit the training data, the inductive bias of the learning algorithm determines which is selected. Correct behaviour is generally under-determined by the training data, so selecting a model with the right behaviour demands inductive bias. It is thus desirable to adapt the training algorithm through fine control over the inductive bias.

Informally, a learning algorithm has a low Vapnik–Chervonenkis (VC) dimension if it can only express simple models. If a learning algorithm has a low VC-dimension then it can be shown that it is likely to generalise well with small amounts of data, regardless of its inductive bias (Vapnik and Chervonenkis 2015). However, with a more expressive learning algorithm (such as DL or ILP) this is insufficient to yield good generalisation, and hence such learning algorithms need a good inductive bias to work well. ILP's strong bias allows it to perform well on small datasets, even though hypotheses can also be highly expressive (Tausend 1994).

The two main components of inductive bias are

- *Limits to the hypothesis space*: Restricting the hypothesis space, i.e. the set of possible output models; and
- *Guiding the search*: The search order for traversing through the hypothesis space, as well as heuristics to assess hypotheses.

Inductive bias in DL. The hypothesis space in DL is largely determined by the network architecture (Szymanski, McCane, and Albert 2018), which we have control over. For example, convolutional neural networks hard-code the assumption that output classes are invariant to shift transformation (Goodfellow, Bengio, and Courville 2016). Train-time methods like dropout and learning rate decay also regularise networks and so add inductive bias (Srivastava et al. 2014). In addition, neural networks have a broad bias towards simplicity, although it is unclear how this bias works (Zhang et al. 2017; Poggio, Liao, and Banburski 2020). The lack of theoretical understanding of DL's search bias implies little explicit control over it.

Inductive bias in ILP. We can restrict the hypothesis space in many ways. A critical design decision for an ILP system is which fragment of FOL represents the examples, background and output model. The classical choice restricts FOL to definite Horn clauses (Muggleton and de Raedt 1994).

In addition, a strong ILP language bias stems from user-supplied constraints on the structure or type of the hypothesis, e.g. mode declarations, meta-rules, or program templates (Payani and Fekri 2019). In some sense these are hyperparameters, as found in any ML system. However, these constraints can be enormously informative, e.g. specifying: which predicates to use in the head or body of the output model; the quantifier of each argument in the predicate; which arguments are to be treated as input and output; and the types of these argument (Evans and Grefenstette 2018).

User-supplied constraints can pertain to (Muggleton and de Raedt 1994) among other things

- Syntax, e.g. second-order schema or bounded term depth;
- Semantics (on the level of terms), e.g. hard-coding the types of predicate arguments, or using determinate clauses; and
- Bounds on the length of the output model.

Two elementary ways to order an ILP search over the set of possible output models are top-down ('from general to specific') or bottom-up ('from specific to general'). At each step of ILP learning, we need a way to score multiple competing hypotheses. This can be done via computing the information gain of the change to the hypothesis (Quinlan 1990) or through probabilistic scoring (Muggleton and de Raedt 1994). A further source of search bias involves specifying the order in which we prune candidate hypotheses.

Comparing ILP with DL. In Table 1 we compare control over inductive bias in ILP and DL. We consider the following, from Witten et al. (2017): language bias (hypothesis space restriction), search bias (how the search through the hypothesis space is ordered), and simplicity bias (how overfitting is prevented).

Bias	**ILP**	**DL**
Simplicity	Bound on program length	Not well understood (besides regularisers e.g. dropout & LR decay)
Language	User constraints, target logic	NN architecture
Search	Search order, hypothesis scoring	Local gradient search

Table 1: Realisations of types of inductive bias

When is control over inductive bias actually hand-coding solutions? The more inductive biases are customised, the more the learning method resembles explicit programming of a solution class. For example, when doing reinforcement learning it is possible to include information about *how* the task should be solved in the reward function. As more information is included, designing the reward function resembles specifying a solution (Sutton and Barto 2018). In ILP, task-specific language biases are often unavoidable for performance reasons, but they risk pruning unexpected solutions, involve a good deal of expert human labour, and can lead to brittle systems which may not learn the problem structure so much as they are passed it to begin with (Payani and Fekri 2019). This problem could be mitigated by progress in automating inductive bias choice in ILP.

Verification of Specifications

In cases where model specification does not give hard guarantees about model behaviour, post-hoc verification is needed. That is, determining, given program M and specification s, whether M's behaviour satisfies s.

Definition 4 (Specification) *Let $\mathcal{L}\colon \mathcal{D} \to \mathcal{M}$ be a learning algorithm. A specification s is a property such that for all models $M \in \mathcal{M}$, the model satisfies the property or not.*

The problem of verifying whether a model satisfies a specification is NP-hard (Clarke and Emerson 1981), both for a neural network (Katz et al. 2017) and for logic programs (Madelaine and Martin 2018).

Verification in ILP. In practice, verifying properties of an output hypothesis is often easy. Suppose you have a propositional theory and want to verify whether this satisfies the specification *False*. This is equivalent to solving satisfiability, and so is at least NP hard. We can verify whether an ILP model M satisfies an arbitrary Datalog specification s by running resolution on $M \cup \{\neg s\}$ to see if it derives False. In fact, this can be done in some cases where s is not in Datalog. For example, this could be done as long as s is in the Bernays-Schönfinkel fragment, albeit in double-exponential time (Piskac, de Moura, and Bjørner 2008). The proof search can be attempted with arbitrary Prolog specifications, but may not terminate.

Verification in DL. To quote Katz et al. (2017), "Deep neural networks are large, non-linear, and non-convex, and verifying even simple properties about them is an NP-complete problem". In practice, complete solvers can verify properties of networks with thousands of nodes, but time out for larger networks (Wang et al. 2018). Incomplete methods can verify properties of networks with $\sim$100 000 ReLU nodes (Singh et al. 2019; Botoeva et al. 2020). Note that the smallest networks that achieve decent results on CIFAR10 have $\sim$50k nodes. Networks can be trained such that they are easier to verify (Xiao et al. 2019).

Post-hoc Model Editing

Definition 5 (Model Editing) *Let $\mathcal{L}\colon \mathcal{D} \to \mathcal{M}$ be a learning algorithm. Let $M \in \mathcal{M}$ be a learned model. Let s be a specification. We apply model editing to M on specification s, if we find a model $M' \in \mathcal{M}$ that has property s without re-applying the learning algorithm $\mathcal{L}$.*

Let d be a distance metric on $\mathcal{M}$. We say that we successfully edit M to fit specification s with respect to distance d if we find a model $M' \in \mathcal{M}$ that has property s and out of all models with property s has minimal distance from M.

Model Editing in ILP. The symbolic representation could make ILP models easier to manipulate than DL models. ILP output models are very interpretable and it is relatively easy for humans to write logical sentences, which should make it in some cases possible to apply model editing.

The output model of ILP is a conjunction of logical clauses. The model can easily be edited, by removing or adding individual clauses. If we simply add clause s to M, then we get a new model $M' = M \cup \{s\}$, which satisfies s and has minimal distance to M with respect to the 'rewrite distance'. When adding clauses, one needs to ensure the new model is still consistent.

A form of post-hoc model editing has been applied to large neural networks, though only by automating the edits. The OpenAI Five agent was trained across several different architectures, with an automatic process for discovering weights to copy (Raiman, Zhang, and Dennison 2019).

Model Editing in DL. After training a large neural network, we (practically speaking) obtain a black-box model. This black-box is not easy to manipulate, owing to the number of parameters and the distributed nature of its representation. Through active learning or incremental learning we can update the model - we could add a module that deals with exceptions, or fine-tune on extra training data for low-performing subgroups. However, these do not give us much control over exactly how the black-box changes (Settles 2009).

Because the black-box is difficult to interpret, we do not fully comprehend what function the network has learned and so are not able to enhance it. Researchers can override the output with a different learned module, but there is no low-level interactive combination of model and human insight.

Transparency

We consider the transparency of learned DL and ILP models, and the transparency of the learning algorithms.

Transparency of the learned model. In contrast to DL models, ILP outputs are represented in an explicit high-level language, making them more transparent. We distinguish between (Lipton 2018): a globally transparent or 'simulatable' model; and a locally transparent or 'decomposable' model.

Decomposability. A decomposable model is one in which each part of the model - each input, parameter, and computation - admits an intuitive explanation, independent of other instances of the model part (Lipton 2018). The many parameters of a neural network form a distributed representation of a nonlinear function, and as such it is unhelpful to reason about individual parameters.

An ILP output model is a conjunction of predicates and literals. When the background is human-specified, each individual predicate will admit an intuitive explanation. When predicates are invented by the ILP system, the results can be counter-intuitive or long; however, they are still themselves encoded as decomposable clauses of intuitive features.

Simulatability. A user can simulate a model if they can take input and work out what the model would output. More precisely, a model M is *simulatable* in proportion to the mean accuracy with which, after brief study of M, a population of users can reproduce the output of M on new data.

A small usability study (n=16) found that access to an ILP program did allow users to simulate a concept they were unable to infer themselves (Muggleton et al. 2018b). It also found, as expected, that increasing the complexity reduced simulatability.

Explainability of the learned model. Since ILP models are relatively transparent, explanations are redundant (except in very large programs). DL explainability is a highly active field of research (Gilpin et al. 2018) and has produced many post-hoc tools, making use of: visualization, text explanations, feature relevance, explanations by example, model simplification and local explanations (Arrieta et al. 2020).

Transparency of the learning algorithm. In DL the learning algorithm optimises weights in a model that already has the same architecture as the output model, such that during training we see many intermediate models. This implies that many of the transparency properties of DL are relevant for its accessibility as well. On the other hand, in ILP we have a distinct training algorithm and output model.

Individual model updates during learning. In DL, backpropagation attributes changes in the loss to individual weights. However, backpropagation can lead to local minima, and so sometimes weights are updated in a direction opposite from the ideal direction. This, along with their (humanly) incomprehensible representation imply that individual updates are not interpretable.

This contrasts with ILP, where each step of the learning algorithm occurs on a symbolic level (for instance, generalising a candidate hypothesis through dropping one literal). In principle a human user could step through ILP learning and understand the concept represented at each step, the complete effect of each change on the model coverage, and the particular data points that constrain the change (though in practice learning can involve many thousands of steps, and so this can take an impractically long time).

Attributing the solution to individual training inputs. Given an ILP output model and an input example, a human can usually assess whether they are consistent. So in ILP it is relatively clear which example or background predicate is causing the ILP algorithm to output a given model.

In DL however, it would be very difficult to (for instance) assess whether an image was a member of the training set of a given model. That is, it is difficult to attribute aspects of the output model to individual inputs.

Inductive bias towards interpretability. User-supplied program constraints and bounds on program length mean that we only generate programs of a certain form, which can be interpretable by construction. Moreover, control over inductive bias itself can be seen as a form of accessibility.

Discussion

We have argued that ILP has a number of safety properties that make it attractive compared to DL:

1. ILP is convenient for *specification*, insofar as it is intuitive to encode examples and properties of correct behaviour;
2. ILP is *robust* to most syntactic changes in inputs;
3. The program templates, and bounds on program length give *control over the inductive bias* in ILP;
4. We can *verify* whether an ILP model satisfies an arbitrary Datalog specification by running resolution;
5. We can *edit* an ILP model by adding or removing clauses;
6. ILP models are *interpretable* as they are quite transparent and are reasonably accessible.

Competitiveness of ILP. It is unlikely that the AI community will adopt ILP if its performance is not competitive. Consider chemistry applications (Srinivasan et al. 1997): ILP continues to be applied (Kaalia et al. 2016), but DL efforts are now more extensive (Cova and Pais 2019). One benchmark is suggestive: ILP found success in the early years of the Comparative Assessment of protein Structure Prediction (Cootes, Muggleton, and Sternberg 2003), but most submissions now use DL (Senior et al. 2019). These results may not be indicative of ILP's current potential as far less research is being invested in ILP than in DL. As a suggestive bound on the ratio of investment, compare the 130 researchers (worldwide) listed on the ILP community hub (Siebers 2019), to the 420 researchers at a single DL lab, Berkeley AI Research, or to the 13,000 attendees of a single DL conference, NeurIPS. This relative neglect might allow for performance gains from research into ILP and hybrid ILP-DL systems.

A deeper concern is the limited domains in which ILP offers its benefits. ILP generates logic programs, where DL approximates continuous functions. We have argued that logic programs are more human interpretable, especially when the predicates used in the program represent concepts we know and use. Our discussion of ILP's transparency only applies to domains where the data is already available on a symbolic level. Moreover, a major theme in recent AI, cognitive science, and linguistics is that rule approaches are insufficient to express or learn most human-level concepts, where continuous features and similarity to exemplars appear necessary (Rouder and Ratcliff 2006; Spivey 2008; Norvig 2012).

Both of the above suggest a need to unify connectionist and symbolic methods. Recent attempts implement relational or logical reasoning on neural networks (Garnelo and Shanahan 2019; Evans and Grefenstette 2018). However, from a safety perspective, these unifications lose desirable properties. We hope that future versions not only increase in performance, but also retain their safety potential.

ILP as specification module. The above suggests a fruitful role for ILP: as a specification generator in a mixed AI system. We may not be able to directly specify safety properties, but may be able to give positive and negative examples of safe behaviour. If it is natural to formulate these examples in natural language or logic, then ILP can generate hypotheses based on these partial specifications. Since ILP output models are easy to interpret, we may be able to verify whether they meet our preferences, and perhaps edit them to account for noisy or missing features. In certain cases (e.g. Datalog), it may be possible to formally verify the hypothesis' correctness. The specification can then be transferred losslessly to any other learning system that can handle logical expressions (e.g. graph neural networks).

ILP's differences with DL suggest solutions to DL's safety shortcomings. We are hopeful that hybrid systems can provide safety guarantees.

Acknowledgements Supported by UKRI studentships EP/S022937/1 and EP/S023356/1 and the AI Safety Research Programme. Thanks to Alex Jackson for suggesting ILP has potential as a specification generator.

References

Anderson, G.; Verma, A.; Dillig, I.; and Chaudhuri, S. 2020. Neurosymbolic Reinforcement Learning with Formally Verified Exploration. In *NeurIPS*.

Armstrong, S.; and Mindermann, S. 2018. Occam's razor is insufficient to infer the preferences of irrational agents. In *NeurIPS*.

Arrieta, A. B.; Rodríguez, N. D.; Ser, J. D.; Bennetot, A.; Tabik, S.; Barbado, A.; García, S.; Gil-Lopez, S.; Molina, D.; Benjamins, R.; Chatila, R.; and Herrera, F. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion* 58.

Botoeva, E.; Kouvaros, P.; Kronqvist, J.; Lomuscio, A.; and Misener, R. 2020. Efficient Verification of ReLU-based Neural Networks via Dependency Analysis. In *AAAI*.

Boytcheva, S. 2002. Overview of ILP Systems. In *Cybernetics and Information Technologies*.

Clarke, E. M.; and Emerson, E. A. 1981. The design and synthesis of synchronization skeletons using temporal logic. In *Workshop on Logics of Programs*.

Condry, N. 2016. Meaningful Models: Utilizing Conceptual Structure to Improve Machine Learning Interpretability. *arXiv:1607.00279* .

Cootes, A. P.; Muggleton, S.; and Sternberg, M. J. 2003. The Automatic Discovery of Structural Principles Describing Protein Fold Space. In *Mol. Biol.*

Cova, T. F.; and Pais, A. A. 2019. Deep Learning for Deep Chemistry: Optimizing the Prediction of Chemical Patterns. *Frontiers in Chemistry* 7.

Cropper, A.; Dumančić, S.; and Muggleton, S. 2020. Turning 30: New Ideas in Inductive Logic Programming. In *IJCAI*.

Cropper, A.; and Morel, R. 2020. Learning programs by learning from failures. *arXiv 2005.02259* .

Cropper, A.; and Tourret, S. 2018. Derivation reduction of metarules in meta-interpretive learning. In *International Conference on Inductive Logic Programming*.

De Raedt, L.; Dries, A.; Thon, I.; Van den Broeck, G.; and Verbeke, M. 2015. Inducing probabilistic relational rules from probabilistic examples. In *IJCAI*.

Edelmann, R.; and Kunčak, V. 2019. Neural-Network Guided Expression Transformation. *arXiv:1902.02194* .

Evans, R.; and Grefenstette, E. 2018. Learning Explanatory Rules from Noisy Data. In *JAIR*.

Garnelo, M.; and Shanahan, M. 2019. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. In *Current Opinion in Behavioral Sciences*.

Gilpin, L.; Bau, D.; Yuan, B.; Bajwa, A.; Specter, M.; and Kagal, L. 2018. Explaining Explanations: An Overview of Interpretability of Machine Learning. In *IEEE DSAA*.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Hüllermeier, E.; Fober, T.; and Mernberger, M. 2013. *Inductive Bias, Encyclopedia of Systems Biology*.

Kaalia, R.; Srinivasan, A.; Kumar, A.; and Ghosh, I. 2016. ILP-assisted de novo drug design. *Machine Learning* 103.

Kashinath, K.; Marcus, P.; et al. 2019. Enforcing Physical Constraints in Neural Neural Networks through Differentiable PDE Layer. In *ICLR DeepDiffEq workshop*.

Katz, G.; Barrett, C.; Dill, D.; Julian, K.; and Kochenderfer, M. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. *Computer Aided Verification. Lecture Notes in Computer Science* 10426.

Kroy, M. 1976. A Partial Formalization of Kant's Categorical Imperative. An Application of Deontic Logic to Classical Moral Philosophy. *Kant-Studien* 67.

Lipton, Z. C. 2018. The mythos of model interpretability. *Queue* 16.

Madelaine, F.; and Martin, B. 2018. On the complexity of the model checking problem. *SIAM J. Comput.* 47.

Márquez-Neila, P.; Salzmann, M.; and Fua, P. 2017. Imposing hard constraints on deep networks: Promises and limitations. *arXiv:1706.02025* .

McNamara, P. 2019. Deontic Logic. https://plato.stanford.edu/archives/sum2019/entries/logic-deontic/.

Mitchell, T. M. 1980. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research.

Muggleton, S. 1999. Inductive Logic Programming: Issues, results and the challenge of Learning Language in Logic. *Artificial Intelligence* 114.

Muggleton, S.; Dai, W.-Z.; Sammut, C.; Tamaddoni-Nezhad, A.; Wen, J.; and Zhou, Z.-H. 2018a. Meta-interpretive learning from noisy images. *Machine Learning* 107.

Muggleton, S.; and de Raedt, L. 1994. Inductive Logic Programming: Theory and methods. *The Journal of Logic Programming* 19/20.

Muggleton, S.; Schmid, U.; Zeller, C.; Tamaddoni-Nezhad, A.; and Besold, T. 2018b. Ultra-Strong Machine Learning: comprehensibility of programs learned with ILP. *Machine Learning* 107.

Norvig, P. 2012. Chomsky and the two cultures of statistical learning. *Significance* 9.

Ortega, P. A.; and Maini, V. 2018. Building safe artificial intelligence: specification, robustness, and assurance. https://medium.com/@deepmindsafetyresearch/building-safe-artificial-intelligence-52f5f75058f1.

Pathak, D.; Krähenbühl, P.; and Darrell, T. 2015. Constrained Convolutional Neural Networks for Weakly Supervised Segmentation. In *IEEE ICCV*.

Payani, A.; and Fekri, F. 2019. Inductive Logic Programming via Differentiable Deep Neural Logic Networks. *CoRR* abs/1906.03523.

Peterson, C. 2014. The categorical imperative: Category theory as a foundation for deontic logic. *Applied Logic* 12.

Piskac, R.; de Moura, L.; and Bjørner, N. 2008. Deciding effectively propositional logic with equality. Technical report, MSR-TR-2008-181, Microsoft Research.

Platt, J. C.; and Barr, A. H. 1988. Constrained differential optimization. In *NeurIPS*.

Poggio, T.; Liao, Q.; and Banburski, A. 2020. Complexity control by gradient descent in deep networks. *Nature Communications* 11.

Powell, D.; and Thévenod-Fosse, P. 2002. Dependability issues in ai-based autonomous systems for space applications. In *IARP-IEEE/RAS joint workshop*.

Quinlan, J. R. 1990. Learning Logical Definitions from Relations. *Machine Learning* 5.

Raiman, J.; Zhang, S.; and Dennison, C. 2019. Neural Network Surgery with Sets. *arXiv:1607.00279* .

Rouder, J. N.; and Ratcliff, R. 2006. Comparing Exemplar- and Rule-Based Theories of Categorization. *Current Directions in Psychological Science* 15.

Senior, A. W.; Evans, R.; Jumper, J.; Kirkpatrick, J.; Sifre, L.; Green, T.; Qin, C.; Žídek, A.; Nelson, A. W. R.; Bridgland, A.; Penedones, H.; Petersen, S.; Simonyan, K.; Crossan, S.; Kohli, P.; Jones, D. T.; Silver, D.; Kavukcuoglu, K.; and Hassabis, D. 2019. Improved protein structure prediction using potentials from deep learning. *Nature* 577.

Settles, B. 2009. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Siebers, M. 2019. inductive-programming community site. https://inductive-programming.org/people.html.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. 2019. An Abstract Domain for Certifying Neural Networks. *ACM Program. Lang.* 3.

Spivey, M. 2008. *The continuity of mind*. Oxford University Press.

Srinivasan, A. 2006. The Aleph System. https://www.cs.ox.ac.uk/activities/programinduction/Aleph/aleph.html.

Srinivasan, A.; King, R. D.; Muggleton, S.; and Sternberg, M. J. E. 1997. Carcinogenesis Predictions Using ILP. In *Inductive Logic Programming, 7th International Workshop*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *JMLR* 15.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *ICLR*.

Szymanski, L.; McCane, B.; and Albert, M. 2018. The effect of the choice of neural network depth and breadth on the size of its hypothesis space. *arXiv:1806.02460* .

Tausend, B. 1994. Representing biases for inductive logic programming. In *ECML*.

Vapnik, V.; and Chervonenkis, A. 2015. *On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities*. Springer.

Wang, S.; Pei, K.; Whitehouse, J.; Yang, J.; and Jana, S. 2018. Efficient Formal Safety Analysis of Neural Networks. In *NeurIPS*.

Witten, I.; Frank, E.; Hall, M.; and Pal, C. 2017. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

Xiao, K. Y.; Tjeng, V.; Shafiullah, N. M.; and Madry, A. 2019. Training for Faster Adversarial Robustness Verification via Inducing ReLU Stability. In *ICLR*.

Yudkowsky, E. 2011. Complex Value Systems are Required to Realize Valuable Futures. The Singularity Institute, San Francisco, CA.

Zhang, C.; Bengio, S.; Hardt, M.; Recht, B.; and Vinyals, O. 2017. Understanding deep learning requires rethinking generalization. In *ICLR*.

Zhang, H.; Chen, H.; Xiao, C.; Gowal, S.; Stanforth, R.; Li, B.; Boning, D.; and Hsieh, C.-J. 2020. Towards Stable and Efficient Training of Verifiably Robust Neural Networks. In *ICLR*.