Differentiable Inductive Logic Programming in High-Dimensional Space

[1]Stanisław J. Purgał*, [2,3]David M. Cerna*, [1]Cezary Kaliszyk*

[1]University of Innsbruck, Innsbruck, Austria
[2]Czech Academy of Sciences Institute of Computer Science (CAS ICS), Prague, Czechia
[3]Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria
stanislaw.purgal@uibk.ac.at, dcerna@{cs.cas.cz, risc.jku.at}, cezary.kaliszyk@uibk.ac.at

Abstract

Synthesizing large logic programs through Inductive Logic Programming (ILP) typically requires intermediate definitions. However, cluttering the hypothesis space with intensional predicates often degrades performance. In contrast, gradient descent provides an efficient way to find solutions within such high-dimensional spaces. Neuro-symbolic ILP approaches have not fully exploited this so far. We propose an approach to ILP-based synthesis benefiting from large-scale predicate invention exploiting the efficacy of high-dimensional gradient descent. We find symbolic solutions containing upwards of ten auxiliary definitions. This is beyond the achievements of existing neuro-symbolic ILP systems, thus constituting a milestone in the field.

Introduction

Neuro-symbolic ILP has become an important topic in the field of inductive synthesis (Cropper and Dumancic 2022) able to automatically find solutions for many inductive learning problems (Evans and Grefenstette 2018). Most approaches use problem-specific restrictions to their respective search space (Shindo, Nishino, and Yamamoto 2021; Sen et al. 2022), what is commonly referred to as *language-bias*. While this is conducive to finding solutions for small to medium problems, synthesizing problems where the shape of the solution may be unknown at the time of running the task is harder, since such space restrictions may remove solutions from the search space.

Predicate Invention (PI) allows one to circumvent the issues associated with restricting the search space. However, in purely symbolic inductive synthesis reliance on large-scale predicate invention is often avoided, as it is too time and space demanding (Muggleton et al. 2012).

In this paper, we present an approach based on differentiable ILP (Evans and Grefenstette 2018) amenable to large-scale PI. Our implementation can synthesize a user-provided number of auxiliary, intensional predicates during the learning process. For most systems, large-scale PI is either intractable due to memory requirements or results in a performance drop as it clutters the hypothesis space. We evaluate the approach on several standard ILP tasks including several for which no neuro-symbolic ILP systems have found solutions so far.

The solutions found by our system, in contrast to the usual ILP solutions, include large numbers of intensional predicates. We posit the usefulness of large-scale PI for the synthesis of complex logic programs. While adding many auxiliary predicates can be seen as a duplication of the search space, and therefore equivalent to multiple initializations of existing neuro-symbolic ILP systems, we demonstrate that our system easily outperforms the re-initialization approach on a particularly hard task.

Furthermore, the proposed approach opens several new interesting research questions including multi-task learning for ILP and knowledge transfer between ILP problems. Finally, we observe, that large-scale invention may hinder our system's ability to learn programs for relatively easy tasks. We provide and test a plausible explanation based on the difficulty of deriving *True* with respect to deriving an explanatory program for the given task.

Related Work

We briefly introduce Inductive logic programming (Cropper and Dumancic 2022), cover aspects of δILP (Evans and Grefenstette 2018) directly relevant to our increase in dimensionality, and compare our approach to related systems inspired by δILP. We assume familiarity with the basics of predicate logic and logic programming, see (Raedt 2008).

Inductive Logic Programming (ILP)

ILP is traditionally a form of symbolic machine learning whose goal is to derive explanatory hypotheses from sets of examples (denoted E^+ and E^-) together with background knowledge (denoted BK). These explanatory hypotheses are largely represented as logic programs of some form (Law, Russo, and Broda 2014; Cropper and Muggleton 2016; Purgal, Cerna, and Kaliszyk 2022; Cropper and Morel 2021; Quinlan 1990). Typically, only a few examples are needed to learn an explanatory hypothesis (Dai et al. 2017).

The most common learning paradigm used by ILP systems is *learning from entailment* (Raedt 2008). The systems referenced above, including δILP, are based on this paradigm which can be stated as follows: A hypothesis H

*These authors contributed equally.

explains E^+ and E^- through the BK, if

$$\forall e \in E^+, BK \wedge H \models e \quad \text{and} \quad \forall e \in E^-, BK \wedge H \not\models e$$

In addition to the learning paradigm, one must also consider how to search through the *hypothesis space*, the set of logic programs constructible using definitions from the *BK* together with the predicates provided as examples. Earlier approaches relied on the *subsumption relation* ($\leq_{sub}$): $H_1 \leq_{sub} H_2 \Rightarrow H_1 \models H_2$ where H_1 and H_2 are plausible hypotheses. Subsumption provides a measure of quality together with a mechanism useful for traversing the hypothesis space starting from the subsumptively most specific, *bottom-up* (Muggleton 1995), or subsumptively most general, *top-down* (Quinlan 1990), hypothesis.

The modern approach to search (Meta-learning) was first implemented as the ILP system *Metagol* (Cropper and Muggleton 2016) using second-order Horn templates to restrict and search the hypothesis space. An example template would be $P(x, y)$:-$Q(x, z), R(z, y)$ where P, Q, and R are variables ranging over predicate symbols. This approach motivated the representation used by δILP and our work.

Differentable ILP

With δILP, *Evans and Grefenstette* developed one of the earliest frameworks providing a differentiable approach to *learning from entailment* ILP. They represented the hypothesis space through a severely restricted form of meta-learning: each template denotes a clause definition with at most 2 literals, and with at most a defined number of *existential*[1] variables per clause (typically ≤ 1). Additionally, each template is associated with a flag denoting whether it is constructed using *BK* only or allows *intensional* definitions to occur (defined by a template). Recursion is also activated by this flag. A predicate definition is a pair of templates where at least one position in the pair is non-empty. The template $((0, false), (1, true))$ accepts (among other clause pairs):

$$p(x, y) : \text{-}succ(x, y)$$
$$p(x, y) : \text{-}succ(x, z), p(x, z)$$

While this definition structure can encode higher arity predicate definitions, learning such predicates is theoretically difficult for this ILP setting (Muggleton, Lin, and Tamaddoni-Nezhad 2015), and thus restricting oneself to dyadic predicates is a common practice.

From a set of predicate definitions (denoted $p_1, \cdots, p_n$) and the *BK*, one can derive a satisfiability problem where each disjunctive clause $C_{i,j}$ denotes the range of possible choices for clause j of predicate definition p_i. The logical models satisfying this formula denote logic programs modulo the definitions and *BK*.

Switching from a discrete semantics over $\{0, 1\}$ to a continuous semantics allows δILP to exploit differentiable logical operators for the construction of a model implementing differentiable deduction. Solving ILP tasks, in this setting, is reduced to minimizing loss through gradient descent.

The input examples E^+, E^- are used by δILP as training data for a binary classifier to learn a model attributing *true* or *false* to ground instances of predicates. This model implements the conditional probability $p(\lambda|\alpha, W, T, L, BK)$, where α is a ground instance, W a set of weights, T the predicate definitions, and L the symbolic language used to describe the problem containing a finite set of atoms.

Each predicate definition $p_i = (t_1^i, t_2^i)$ is associated with a weight matrix whose shape is $d_1 \times d_2$ where d_j denotes the number of clauses constructible using the BK and L modulo the constraints of t_j^i. This implies that the number of weights is *quartic* in the number of predicate definitions (as each definition uses 2 clauses and each clause uses 2 body predicates). Each weight denotes δILP's confidence in a given pair of clauses correctly defining the predicate p_i. This construction of the weight tensor is referred to as *splitting per definition* and will be discussed in **Contributions**.

δILP implements differentiable inferencing by providing each clause c with a function $f_c : [0, 1]^m \to [0, 1]^m$ whose domain and range are valuations of grounded predicates. Note, m is not the number of predicates, rather it is the number of groundings of every predicate, a much larger number. Consider a predicate definition p_1 admitting the clause pair (c_1, c_2), and let the current valuation be $\mathcal{EV}_i$ and $g : [0, 1] \times [0, 1] \to [0, 1]$ a function computing $\vee$-*clausal* (disjunction between clauses). Assuming we have a definition of f_c, then $g(f_{c_1}(\mathcal{EV}_i), f_{c_2}(\mathcal{EV}_i))$ denotes one step of *forwards-chaining*. Computing the weighted average (denoted by $\otimes$) overall clausal combinations admitted by p_i, using the *softmax* of the weights, then summing these values, and finally performing $\vee$-*step* (disjunction between inference steps) between their sum and $\mathcal{EV}_i$ results in $\mathcal{EV}_{i+1}$. This is repeated n times (the desired number of forward-chaining steps), where $\mathcal{EV}_0$ is derived from the BK.

The above construction is still dependent on a precise definition of f_c. Let $c_g = p(x_1, x_2) : \text{-}Q_1(y_1, y_2), Q_2(y_3, y_4)$ where $y_1, y_2, y_3, y_4 \in \{x_1, x_2, z\}$ and z is an existential variable. We want to collect all ground predicates p_g for which a substitution θ into $Q_1, Q_2, y_1, y_2, y_3, y_4$ exists such that $p_g \in \{Q_1(y_1, y_2)\theta, Q_2(y_3, y_4)\theta\}$. These ground predicates are then paired with the appropriate grounding of the lefthand side of c_g. The result of this process can be reshaped into a tensor emphasizing which pairs of ground predicates derive various instantiations of $p(x_1, x_2)$. In the case of existential variables, there is one pair per atom in the language. Pairing this tensor with some valuation $\mathcal{EV}_i$ allows one to compute $\wedge$-*literal* (conjunction between literals of a clause) between predicate pairs. As a final step, we compute $\vee$-*exists* (disjunction between variants of literals with existential variables) between the variants, and thus computed the tensor required for a step of *forward-chaining*.

The above process is parameterized by four differentiable operations for conjunction and disjunction. we will leave further discussion of these operators to **Methodology**.

Related Approaches

To the best of our knowledge, the *Logical Neural Network* (LNN) based ILP system presented in (Sen et al. 2022) is the most recent systems to build on δILP. They use templating, but only to learn non-recursive *chain rules*. Note, this is easily simulated using δILP templates, especially for learning

[1] Not present in the head of the clause.

the short rules presented by the authors. Their investigation focused on particular parameterized, differentiable, logical operators optimizable for ILP. Earlier investigations, such as NeuraILP (Yang, Yang, and Cohen 2017), experimented with various *T-norms* as differentiable operators, and part of the investigation reported in (Evans and Grefenstette 2018) studied their influence on learning. The authors leave scaling their approach to larger, possibly recursive programs as future work, a limitation addressed herein.

In (Shindo, Nishino, and Yamamoto 2021), the authors built upon δILP, but further restrict templating to add a simple term language (at most term depth 1). Thus, even under the severe restriction of at most one body literal per clause, they can learn predicates for list *append* and *delete*. Nonetheless, scalability remains an issue. *Neural Logical Machines* (Dong et al. 2019), in a limited sense, addressed the scalability issue. The authors modeled propositional formulas using multi-layer perceptrons wired together to form a circuit. This circuit is then trained on many (10000s) instances of a particular ILP task. While the trained model was accurate, interpretability is an issue, as it is unclear how to extract a symbolic expression. Our approach provides logic programs as output, similar to δILP.

Some related systems loosely related to our work are *Logical Tensor Networks* (Donadello, Serafini, and d'Avila Garcez 2017), *Lifted Relational Neural Networks* (Sourek et al. 2017), *Neural theorem prover* (Minervini et al. 2020), and *DeepProbLog* (Manhaeve et al. 2021). While some, such as Neural theorem prover, can learn rules, it also suffers from scaling issues. Overall, these systems were not designed to address learning in an ILP setting.

Contributions

The space complexity of δILP is roughly *quintic* in the number of predicate symbols. As a result, experiments performed in (Evans and Grefenstette 2018) used task specific predicate definitions and templates, thus incurring a strong *language bias*. This is evident in their experiments where for certain task, i.e. the *even* predicate (See Figure 1), multiple results were listed, with different templates used. Their restrictions force only a few of the many possible solutions to be present in the search space. By switching the templates the learnable predicates change.

The necessity of these restrictions partially entails the author's choice to split programs *per definition* and assign weights to each instance resulting in a very large vector v of learnable parameters. While this implies high dimensionality, as discussed above, *softmax* is applied to v during differentiable inferencing, thus turning v into a distribution and logarithmically reducing its dimensionality.

Our investigation aims to increase the dimensionality of the search space while maintaining the efficacy of the differentiable inferencing implemented in δILP. We proceed by adding many auxiliary predicate definitions with uniformly defined templates, i.e. ((1,true),(1,true)). This largely reduces the language biases mentioned previously. Though given the space complexity of the δILP, doing so is highly intractable, thus amending the procedure outlined in the previous section is necessary. The weight tensor heavily influences the computational and space complexity of δILP. A design choice made in (Evans and Grefenstette 2018) was to assign weights to each possible pair of clauses, thus splitting programs by predicates definitions (*per definition*). This design choice is discussed in Appendix F of their paper. The authors attempted to split *per clause*, however, that approach was "incapable of escaping local minima on harder tasks".

In our work, we split *per literal*. This results in a significant reduction in space complexity, *quintic* to *quadratic*, which we exploit by adding a multitude of auxiliary predicated definitions. An illustration of the different splits can be found in Figure 2. We refer to this new system as δILP_2 which we test using the following five hypotheses that concern either increased dimensionality or aspects of the abovementioned design choices (see **Experiments**):

Hypothesis 1 Adding auxiliary predicate definitions aids the search process.

Hypothesis 2 Adding auxiliary predicate definitions is not the same as re-initialization of weights.

Hypothesis 3 Derivability of **True** from the background knowledge negatively influences accuracy and correctness.

Hypothesis 4 Adding auxiliary predicate definitions while learning a program simultaneously with a necessary subprocedure improves accuracy and correctness.

Hypothesis 5 Adding auxiliary predicate definitions while learning a program and pre-initializing some of the definitions with the solution to a necessary sub-procedure improves accuracy and correctness.

Methodology

Our approach deviates from δILP methodologically concerning how differential inferencing is implemented, the use of differentiable logical operators, how the quality of training outcomes are measured, and how training examples are batched. We cover these differences below.

Experimental Setup

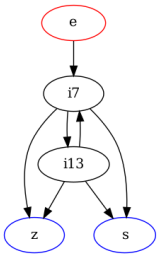
The Hyperparameter settings used for all experiments are:

- using *Adam* (Kingma and Ba 2015) optimizer
- learning rate of 0.05
- weights initialization distribution is uniform in range $(0, 10)$.
- runs end early when the training loss is below 10^{-3}

We ran our experiments on a computational cluster with 16 nodes, each with 4 GeForce RTX 2070 (with 8 GB of RAM) GPUs, running Linux 4.15.0. Most tasks used a single GPU, and some more memory-hungry ones used 4. All our experiments are reported over 100 runs using different pseudo-random seeds unless otherwise stated.

We developed our modified version of δILP using PyTorch (Paszke et al. 2019) (version 1.11) implementing the changes discussed below and a few technical optimizations.

```
1  e(A,B):-i7(B,B),i7(B,B)
2  e(A,B):-i7(B,A),i7(A,B)
3  i7(A,B):-z(C,B),z(C,B)
4  i7(A,B):-i13(A,C),s(C,B)
5  i13(A,B):-s(C,B),i7(C,C)
6  i13(A,B):-z(A,B),z(C,A)
```




```
1  e(A,B):-i2(C,B),i2(C,B)
2  e(A,B):-i18(B,C),i2(C,A)
3  i2(A,B):-i2(C,C),i18(B,C)
4  i2(A,B):-z(A,B),z(A,B)
5  i18(A,B):-z(C,A),z(A,A)
6  i18(A,B):-s(B,C),s(C,A)
```

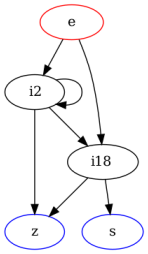


Figure 1: Sample *even* solutions using 20 predicate definitions, trimmed to used definitions

$$[\texttt{gp}(A,B)\texttt{ :- i1}(A,C),\texttt{i1}(C,B)] \qquad [\texttt{gp}(A,B)\texttt{ :- i1}(A,C),\texttt{i1}(C,B)] \qquad \texttt{gp}(A,B)\texttt{ :- }[\texttt{i1}(A,C)],[\texttt{i1}(C,B)]$$

$$\left[\begin{array}{l}\texttt{i1}(A,B)\texttt{ :- mom}(A,B),\texttt{mom}(A,B)\\ \texttt{i1}(A,B)\texttt{ :- dad}(A,B),\texttt{dad}(A,B)\end{array}\right] \qquad \begin{array}{l}[\texttt{i1}(A,B)\texttt{ :- mom}(A,B),\texttt{mom}(A,B)]\\ [\texttt{i1}(A,B)\texttt{ :- dad}(A,B),\texttt{dad}(A,B)]\end{array} \qquad \begin{array}{l}\texttt{i1}(A,B)\texttt{ :- }[\texttt{mom}(A,B)],[\texttt{mom}(A,B)]\\ \texttt{i1}(A,B)\texttt{ :- }[\texttt{dad}(A,B)],[\texttt{dad}(A,B)]\end{array}$$

Figure 2: Splits of the "grandparent" predicate definition – per definitions, clauses, and literals (denoted by [ ])

T-norm for fuzzy logic

As discussed in **Related Work**, both δILP and our approach require four differentiable logic operators to perform differentiable inferencing. The choice of these operators greatly impacts overall performance. The Author's of δILP experimented with *t-norms* or (*triangle norms*), continuous versions of *classical* conjunction (Esteva and Godo 2001) from which continuous versions of other logical operator are constructed. The standard t-norms are *max* ($x \wedge y \equiv \max\{x, y\}$), *product* ($x \wedge y \equiv x \cdot y$) and Łukasiewicz ($x \wedge y \equiv \max\{x+y-1, 0\}$). For simplicity we refer to all operators derived from a t-norm by the conjunctive operator, i.e. $x \vee y \equiv \min\{x, y\}$ is referred to as *max* when discussing the chosen t-norm.

	δILP	δILP$_2$
$\wedge$-Literal	product	product
$\vee$-Exists	max	max
$\vee$-Clausal	max	max
$\vee$-Step	product	max

When computing a large number of inference steps *product* produces vanishingly small gradients. We require computing more inference steps as we are producing much larger programs (See Figure 7). Thus, we use *max* for $\vee$-step.

Adjustments to Differentiable Inferencing

The authors of δILP process predicate definitions individually in a loop when computing gradients. Instead we operate on a single tensor (*truth tensor*). A similar approach was used by the (Dong et al. 2019). Figure 3 illustrates how we perform differentiable inferencing using truth tensors. Below we describe the process.

We start by expanding the previously computed *truth tensor* to construct the *adjusted tensor* which stores an expansion of each predicate by the nine variable choices (which of the 3 variables to use for which of the 2 arguments). This is analogous to creating nine new tertiary predicates from each binary predicate (see Figure 4). Next, for each literal (two per clause) in each clause (two per definition) we compute a weighted average of the truth tensor, the so-called *weighted tensor*. Averaging may result in numerical instability and thus values slightly greater than one may be computed. This can result in negative loss or *nan* errors. We deal with this problem by rounding without modifying gradients.

We reduce the *weighted tensor* to the *valuation tensor* through application of the various differentiable logic operators, i.e. we perform $\wedge$-literals, $\vee$-exists, and $\vee$-clausal. To construct the next truth tensor we apply $\vee$-step between the previous truth tensor and the *valuation tensor*.

The overall complexity of our method (both space and time) is determined by the maximum size of the truth tensors computed during differentiable inferencing (inference steps $\times$ predicates^2 $\times$ atoms^3). Both computation cost and memory usage scale with inference step count since all values are needed to properly perform *backpropagation*.

Considered outcomes

Adding auxiliary predicate definitions and simplified templating entails additional training outcomes.

Overfitting The experimental design presented by the authors of δILP avoids overfitting as the search space did not include programs that fit the training data and are not correct solutions. We allow for a much larger search space that contains such programs.

For example, when learning to recognize even numbers, with enough invented predicates, it is possible to remember all even numbers provided in E^+. Thus, we add a validation step testing our solutions on unseen data (numbers up to 20 after training on numbers up to 10).

We have observed that δILP$_2$ rarely overfits, even when it clearly could. A plausible explanation is that shorter, precise solutions have a higher frequency in the search space.

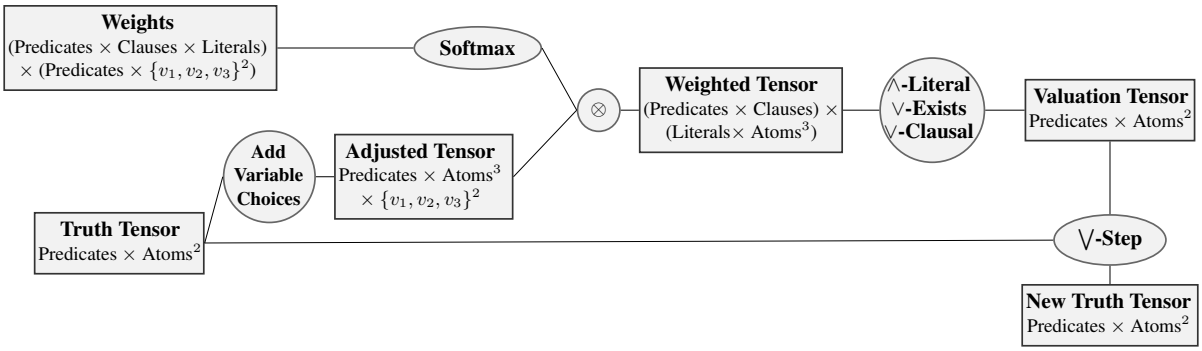


Figure 3: Differentiable inferencing within δILP$_2$

p_1(A,B,C) :- p(A,A) p_2(A,B,C) :- p(A,B) p_3(A,B,C) :- p(A,C)
p_4(A,B,C) :- p(B,A) p_5(A,B,C) :- p(B,B) p_6(A,B,C) :- p(B,C)
p_7(A,B,C) :- p(C,A) p_8(A,B,C) :- p(C,B) p_9(A,B,C) :- p(C,C)

Figure 4: Extension of a binary predicate *p* to 9 tertiary predicates, one for each variable choice

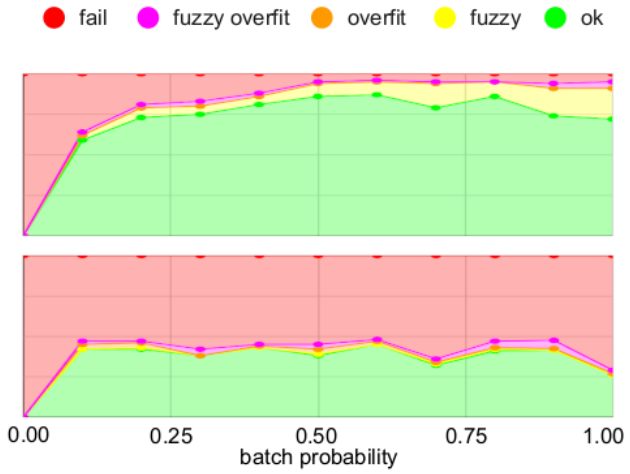


Figure 5: Batch Probability: *fizz* (Top) and *member* (Bottom)

Fuzzy solutions Another class of solutions – this one occurring in both our approach and δILP – are fuzzy solutions. The resulting model correctly predicts the answer using fuzzy logic, but when collapsed to a classical logic program (using the predicate with the highest weight) the program is incorrect. Typically, fuzzy solutions are worse at generalizing – they are correct when tested using the training parameters (for example, inference steps), and break on unseen input. Fully correct solutions for *even* are translatable into a program correct for all natural numbers, a fuzzy solution would be wrong beyond a certain point.

Batch probability

Our approach requires computing values for all predicates over all combinations of atoms. For this reason, it does not make much sense to use a typical approach to mini-batching in this scenario. Instead of parameterization by *batch size*, we use a *batch probability* – the likelihood of an example contributing to gradient computation. We experimented with various probabilities and their influence on the outcome.

Half the loss is computed using E^+ and half using E^-. Regardless of the chosen examples, the loss is kept balanced. If no examples from E^+ (E^-) are chosen, we set that half of the loss to 0 (with 0 gradient). Figure 5 illustrates influence of *batch probability* on performance which degrades only in the vicinity 0.0 and 1.0. In all other experiments, we used the default value of 0.5.

task	predicates definitions	original δILP result	our results			
		fuzzily correct	all correct	fuzzily correct	correct on training	fuzzily correct on training
predecessor	1	**100**	100	**100**	100	100
even / odd	2	**100**	88	**99**	88	99
even / +2	2	48.5				
less than	1	**100**	44	44	48	48
fizz*	3	10	88	**98**	88	98
buzz* w. +2 & +3	2	14	99	**100**	99	100
buzz* w. only +1	4	0#	61	**66**	61	67
member	1	**100**	23	24	28	32
length	2	**92.5**	24	24	25	30
grandparent	2	**96.5**	41	42	90	91
undirected edge	1	**100**	100	**100**	100	100
adjacent to red	2	50.5	97	**99**	97	99
two children	2	95	60	**100**	60	100
graph colouring	2	**94.5**	73	74	99	100
graph colouring*	2	94.5	97	**98**	98	100
connectedness	1	**100**	40	40	91	91
cyclic	2	**100**	30	31	97	100

Table 1: Comparison with δILP results as reported by (Evans and Grefenstette 2018) – except results marked with #, computed by us

Experiments

We compare our approach to δILP using tasks presented in (Evans and Grefenstette 2018). The results are shown in Table 1. We used an additional 50 predicate definitions (1 for the target), 25 inference steps, batch probability 0.5, 5000 gradient descent steps, and ran each task 100 times. Both the number of intensional predicate definitions and inference steps used are beyond the capabilities of δILP. Also, it must be emphasized that the authors of δILP defined success as a low loss on the training data ($1e-04$). In our setting, this translates to *fuzzily correct on training*. Given that δILP was unable to overfit, we classify its results as *fuzzily correct*.

δILP experiments considered simple family relations

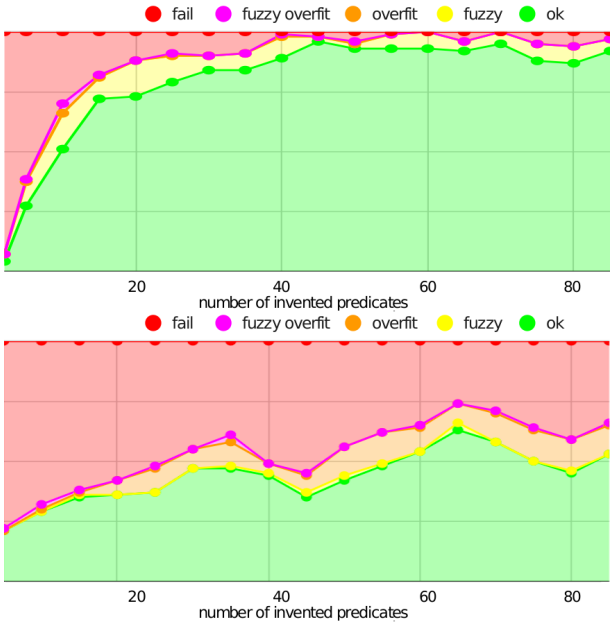


Figure 6: Results of changing the number of invented predicates – in *fizz* (top) and *less than* (bottom). Runs used 20 inference steps, batch probability 0.5, and 5k gradient descent steps. The number of intensional predicates ranged between 1 and 85.

tasks. We skipped the majority of these tasks for the following reason: the solution programs were not strictly correct. For example, a solution for *son* required the son have a sibling or a child of his own. This makes validation difficult as proper validation would mean no solutions learned.

We outperform δILP on many of the harder tasks presented in (Evans and Grefenstette 2018). However, some of the simpler tasks presented an unexpected challenge. To clarify, let us first consider tasks that require a single predicate definition. In such cases, the design choice to split *per definition* results in a very roundabout brute-force search.

For a more complex task such as *length*, the precise cause of the performance drop when using δILP_2 is unclear. Given our confirmation of **Hypothesis 3**, we posit a plausible explanation via the influence of derivability of **True**. Though, the precise reasons for this performance drop remain open.

Some tasks, as defined in (Evans and Grefenstette 2018), use excessively sparse training data. This was sufficient for δILP as overfitting was not a possibility. For example, when learning *buzz* (divisibility by 5), only numbers up to nine were used; this entails that any program accepting only 0 and 5 would be considered correct. For such cases, we use slightly more training data, i.e. including ten as a positive example (such tasks are marked with * in Table 1) or underline the success rate of over-fitting. We continue this section by addressing each of the hypotheses listed in **Contributions**.

solution type	Truths	Truths & One-shots	One-shots	Std. *BK*
correct	6%	13%	32%	45%
fuzzy correct	30%	29%	46%	66%
correct on training	6%	13%	32%	45%
fuzzy correct on training	30%	32%	47%	67%

Table 2: Influence of *truths* and *one-shots* on accuracy.

task	all correct	fuzzily correct	correct on training	fuzzily correct on training
divisible by 2	88	99	88	99
divisible by 3	88	98	88	98
divisible by 6	60	65	60	65
all together	47	66	47	68

Table 3: Comparison of learning divisibility tasks separately and together.

Hypothesis 1

In addition to Table 1, We provide the results of experiments run using the *fizz* task and *lessthan* task where we vary the number of predicate definitions used. Each run uses 25 inference steps, a batch probability of 0.5, and 5000 gradient descent steps (see Figure 6). In total, we performed 100 runs using up to 85 intensional predicates.

Hypothesis 2

To illustrate that using a multitude of predicate definitions is not equivalent to re-initialization of weights we ran the *buzz* task with a *BK* containing only *successor* and *zero*. The authors of δILP did not attempt this task. As shown in Table 1, δILP struggles to find fuzzily correct solutions. We ran δILP_2 10000 times using four predicate definitions. It only learned two fuzzily correct solutions. To achieve a 60% chance of a single successful run, when limiting intensional predicates to four, requires over 5000 runs, and thus provides strong support for our hypothesis.

Hypothesis 3

During our experiments, we noticed a tendency for δILP_2 to use literals of the form $p(z, z)$ where z is existential; this typically entailed failure. Furthermore, introduction of atomic predicate definitions $q(a, a)$ (*one-shots*) into the *BK* such that no other predicate in the *BK* uses the atom a results in δILP_2 using this predicate often to derive a *truth* predicate (a predicate which holds for all atoms), and in doing so, failing to learn. We tested our hypothesis using *even* while limiting ourselves to five predicate definitions (Which makes it harder to learn a solution and thus emphasizes the loss in performance). In addition we introduced *one-shots* and/or *truths* into the *BK*. The results are found in Table 2. Observe the decrease in accuracy when these pathological predicates are added to the *BK* with *truths* being the most damaging, thus supporting our hypothesis.

Hypothesis 4

Introducing many auxiliary predicate definitions allows one to learn multiple tasks simultaneously and potentially bene-

```
d6(A,B):-i37(B,C),d3(A,B)
d6(A,B):-s(B,A),s(B,A)
d3(A,B):-i27(B,B),i27(A,B)
d3(A,B):-i24(C,B),i48(A,C)
d2(A,B):-z(C,C),i41(C,B)
d2(A,B):-i44(B,B),i44(B,B)
i20(A,B):-s(C,B),i44(C,C)
i20(A,B):-z(B,A),z(A,C)
i24(A,B):-s(B,C),z(B,B)
i24(A,B):-i38(C,A),s(C,A)
i27(A,B):-z(B,A),z(C,A)
```

```
i27(A,B):-s(A,B),s(A,B)
i37(A,B):-z(B,A),z(B,A)
i37(A,B):-i20(A,C),s(C,A)
i38(A,B):-z(B,B),z(A,B)
i38(A,B):-d3(C,C),s(C,A)
i41(A,B):-z(B,A),z(A,B)
i41(A,B):-i41(A,C),z(B,B)
i44(A,B):-s(C,A),i20(B,C)
i44(A,B):-z(C,B),z(B,B)
i48(A,B):-s(C,B),s(B,A)
i48(A,B):-z(A,B),z(A,B)
```

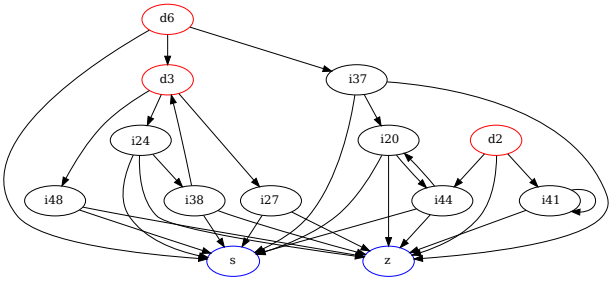


Figure 7: Above: Definitions of used intensional predicates when learning *div6*, *div3* and *div2* together. Below: Call structure of solution when learning *div6*, *div3* and *div2*.

fit from *knowledge transfer*. Our test required learning three predicates: divisibility by 2, 3, and 6 – both separately and all at the same time.

The results are shown in Table 3 and suggest that the probability of solving all three problems together is lower than solving *divisible by 6* on its own. It is about equal to the probability of solving all of the tasks separately ($88\%{\cdot}88\%{\cdot}60\% \approx 46\%$), suggesting no benefit from knowledge reuse. Though, we do see *knowledge transfer* in some output programs, see Figure 7, where *div6* predicate calls *div3* directly and indirectly uses a predicate used by *div2*.

Hypothesis 5

Adding auxiliary predicate definitions allows one to initialize some of the weights using weights learned to solve a related task. For example, *fizz* and *even* have similar solutions. We tested this hypothesis by initializing some of the weights associated with auxiliary predicate definitions by those used by the solution to *even* when learning *fizz* and vice versa. While initializing with *fizz* and learning *even* shows a slight improvement in accuracy (91% correct solutions), the other direction resulted in a decreased accuracy (72% correct solutions). There are various solutions to *even* and some of these do not coalesce as well with solutions to *fizz* possibly explaining the lower performance. In our experimental setting, solutions to fizz require learning $p(x, y) : -succ(x, z), succ(z, y)$, a very useful predicate for the *even* task, however, solutions to *even* which learn *odd* are not beneficial to learning *fizz*.

Conclusion & Future Work

The main contribution of this work (**Hypothesis** 1) is providing strong evidence that additional predicate definitions (beyond what is necessary) improve performance. Verification of this hypothesis used δILP_2, our modified version δILP. Additionally, we verified that the performance gains were not simply due to properties shared with weight reinitialization (**Hypothesis** 2).

However, performance gains were not uniform over all tasks investigated, as for some tasks (*length*), δILP outperformed our modification. While we do not have a definitive theory concerning this anomaly, our verification of **Hypothesis** 3 provides a plausible explanation. Predicates that are true for all atoms seem to produce local minima which cannot be easily escaped. How difficult it is to derive **True**, with respect to learning a correct program, seems to have consequences on performance.

δILP_2 provides the opportunity to test knowledge transfer through multi-task learning (**Hypothesis** 4) and weight transfer (**Hypothesis** 5). Though, we did not observe statistically significant changes in performance. We did see a slight benefit when initializing with *fizz* and learning *even*, and as pointed out in Figure 7, we did see reuse of discovered solutions.

As a continuation of our investigation, we plan to integrate ILP with Deep Neural Networks as a hybrid system, that is trainable end-to-end through backpropagation. The first steps were presented in (Evans and Grefenstette 2018). One can imagine the development of a network inferring a discrete set of objects in an image (Carion et al. 2020), or integration with Transformer-based (Vaswani et al. 2017) language models that produce atoms δILP_2 can process. This can lead to a network that responds to natural language queries based on a datalog database.

We also consider further optimization of inferencing within δILP_2. By stochastically using only a fraction of all clauses matching a template, similar to the REINFORCE algorithm (Williams 1992), there is the potential to save a significant amount of computation time and memory. Thus, we can further exploit the benefits of adding auxiliary predicate definitions and possibly solve more complex tasks. A similar idea would be to split a program into a number of small parts and learn by gradient descent in a different programming paradigm, for example, functional programming.

Finally, our approach may be improved by adding an auxiliary loss. Such loss could range from measuring the used program size (to promote smaller, more general solutions) to using a language model to guide the search towards something that looks like a solution.

Acknowledgements

Supported by the ERC starting grant no. 714034 SMART, the Math$_{LP}$ project (LIT-2019-7-YOU-213) of the Linz Institute of Technology and the state of Upper Austria, Cost action CA20111 EuroProofNet, and the Doctoral Stipend of the University of Innsbruck.

References

Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; and Zagoruyko, S. 2020. End-to-End Object Detection with Transformers. *CoRR*, abs/2005.12872.

Cropper, A.; and Dumancic, S. 2022. Inductive Logic Programming At 30: A New Introduction. *J. Artif. Intell. Res.*, 74: 765–850.

Cropper, A.; and Morel, R. 2021. Learning programs by learning from failures. *Mach. Learn.*, 110(4): 801–856.

Cropper, A.; and Muggleton, S. 2016. Metagol System. github.com/metagol/metagol.

Dai, W.; Muggleton, S. H.; Wen, J.; Tamaddoni-Nezhad, A.; and Zhou, Z. 2017. Logical Vision: One-Shot Meta-Interpretive Learning from Real Images. In Lachiche, N.; and Vrain, C., eds., *ILP 2017*, volume 10759 of *LNCS*, 46–62. Springer.

Donadello, I.; Serafini, L.; and d'Avila Garcez, A. S. 2017. Logic Tensor Networks for Semantic Image Interpretation. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 1596–1602. ijcai.org.

Dong, H.; Mao, J.; Lin, T.; Wang, C.; Li, L.; and Zhou, D. 2019. Neural Logic Machines. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Esteva, F.; and Godo, L. 2001. Monoidal t-norm based logic:towards a logic for left-continuous t-norms. *Fuzzy Sets and Systems*, 124(3): 271–288. Fuzzy Logic.

Evans, R.; and Grefenstette, E. 2018. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61: 1–64.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980.

Law, M.; Russo, A.; and Broda, K. 2014. Inductive Learning of Answer Set Programs. In *Proceedings of Logics in Artificial Intelligence*, 311–325. Springer.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and Raedt, L. D. 2021. Neural probabilistic logic programming in DeepProbLog. *Artif. Intell.*, 298: 103504.

Minervini, P.; Bosnjak, M.; Rocktäschel, T.; Riedel, S.; and Grefenstette, E. 2020. Differentiable Reasoning on Large Knowledge Bases and Natural Language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 5182–5190. AAAI Press.

Muggleton, S. 1995. Inverse Entailment and Progol. *New Generation Computing*, 13(3&4): 245–286.

Muggleton, S. H.; Lin, D.; and Tamaddoni-Nezhad, A. 2015. Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. *Mach. Learn.*, 100(1): 49–73.

Muggleton, S. H.; Raedt, L. D.; Poole, D.; Bratko, I.; Flach, P. A.; Inoue, K.; and Srinivasan, A. 2012. ILP turns 20 - Biography and future challenges. *Mach. Learn.*, 86(1): 3–23.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

Purgal, S. J.; Cerna, D. M.; and Kaliszyk, C. 2022. Learning Higher-Order Programs From Failures. In *IJCAI-22*, volume abs/2112.14603. Accepted, to appear.

Quinlan, J. R. 1990. Learning Logical Definitions from Relations. *Mach. Learn.*, 5: 239–266.

Raedt, L. D. 2008. *Logical and relational learning*. Cognitive Technologies. Springer. ISBN 978-3-540-20040-6.

Sen, P.; de Carvalho, B. W. S. R.; Riegel, R.; and Gray, A. G. 2022. Neuro-Symbolic Inductive Logic Programming with Logical Neural Networks. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, 8212–8219. AAAI Press.

Shindo, H.; Nishino, M.; and Yamamoto, A. 2021. Differentiable Inductive Logic Programming for Structured Examples. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, 5034–5041. AAAI Press.

Sourek, G.; Svatos, M.; Zelezný, F.; Schockaert, S.; and Kuzelka, O. 2017. Stacked Structure Learning for Lifted Relational Neural Networks. In Lachiche, N.; and Vrain, C., eds., *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers*, volume 10759 of *Lecture Notes in Computer Science*, 140–151. Springer.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. *CoRR*, abs/1706.03762.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.*, 8: 229–256.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In Guyon, I.; von Luxburg, U.; Bengio, S.; Wallach, H. M.; Fergus, R.; Vishwanathan, S. V. N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2319–2328.