

Differentiable Inductive Logic Programming in High-Dimensional Space

Stanisław J. Purgal^{a,*}, David Cerna^b and Cezary Kaliszyk^a

^aUniversity of Innsbruck

^bCzech Academy of Sciences Institute of Computer Science
ORCID ID: David Cerna <https://orcid.org/0000-0002-6352-603X>,
Cezary Kaliszyk <https://orcid.org/0000-0002-8273-6059>

Abstract. Synthesizing large logic programs through symbolic Inductive Logic Programming (ILP) typically requires intermediate definitions. However, cluttering the hypothesis space with intensional predicates typically 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 extending the δ ILP approach to inductive synthesis with large-scale predicate invention, thus allowing us to exploit the efficacy of high-dimensional gradient descent. We show that large-scale predicate invention benefits differentiable inductive synthesis through gradient descent and allows one to learn solutions for tasks beyond the capabilities of existing neuro-symbolic ILP systems. Furthermore, we achieve these results without specifying the precise structure of the solution within the *language bias*.

Introduction

Neuro-symbolic ILP is quickly becoming one of the most important research domains in inductive synthesis [3]. Systems such as δ ILP can consistently learn solutions for many inductive learning problems [11, 25]. Nonetheless, searching through the hypothesis space remains a challenging task. To deal with this difficulty, inductive learners introduce problem-specific restrictions that reduce the size of the respective search space [26, 24]; what is commonly referred to as *language bias*. While this is conducive to solving simple learning tasks, more complex synthesis tasks or tasks where the required language bias is not easily specifiable remain a formidable challenge.

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 time- and space-wise too demanding [18], and has only been used effectively in very restricted settings [19].

This paper presents an approach based on *differentiable* ILP [11], a very influential approach to neuro-symbolic inductive synthesis, amenable to large-scale predicate invention. Our implementation can synthesize a user-provided number of auxiliary, *intensional* predicates during the learning process. Large-scale predicate invention is either intractable for most systems due to memory requirements or results in a performance drop as it clutters the hypothesis space. In contrast, gradient descent methods generally benefit from large search space (high dimensionality). Thus, we propose introducing a

large number of templates to improve performance. We evaluate the approach on several standard ILP tasks (many derived from [11]), including several which existing neuro-symbolic ILP systems find to be a significant challenge (see *Hypothesis 1*).

The solutions found by our extension of δ ILP, in contrast to the usual ILP solutions, include large numbers of intensional predicates. We posit the usefulness of large-scale predicate invention for synthesizing 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 extension of δ ILP easily outperforms the re-initialization approach on a particularly challenging task (see *Hypothesis 2*). We compare to the most relevant existing approach, δ ILP (presented in [11]).

Unlike the experiments presented in [11], which specify the solution’s precise *template* structure, we use generic templates. Thus, our experiments force the learner to find the correct predicates for the given task and the correct structure of the solution. In this more complex experimental setting, our approach performs as well as δ ILP, and for some of the more challenging tasks, outperforms the earlier system. In particular, we outperform δ ILP on tasks deemed difficult in [11] such as *fizz*, *buzz*, and *even*.

Furthermore, we propose an adjusted measure of task difficulty. It was proposed in [11] that the number of *intensional* predicates is a good measure of learning complexity. While our results do not contradict this assertion, it is more precise to talk about *intensional* predicates that relate two variables through a third existentially quantified variable; our system swiftly illustrates this by solving *buzz* (requiring four intensional predicates) consistently, but performing poorly on the seemingly simpler task of computing $Y = X + 4$ that requires only two intensional predicates (Hypothesis 3). Understanding what makes a task difficult will aid future investigations in developing improved approaches to neuro-symbolic inductive synthesis.

In this paper, we present (i) an extension of δ ILP capable of large-scale predicate invention, (ii) we experimentally show that our extension outperforms δ ILP on challenging tasks when *language bias* is reduced, (iii) we experimentally show that large-scale predicate invention differs from weight re-initialization in the differentiable inferencing setting, (iv) we propose a novel criterion for task complexity and experimentally validate it.

* Corresponding Author. Email: sjpurgal@protonmail.ch

Related Work

We briefly introduce Inductive logic programming [3], cover aspects of δ ILP [11] directly relevant to our increase in dimensionality, and compare our approach to related systems inspired by δ ILP. We assume familiarity with basic logic and logic programming; see [23].

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). Investigations often represent explanatory hypotheses as logic programs of some form [4, 5, 13, 21, 22]. A benefit of this approach is that only a few examples are typically needed to learn an explanatory hypothesis [7].

The most common learning paradigm implemented within ILP systems is *learning from entailment* [23]. The systems referenced above, including δ ILP, use this paradigm which is succinctly stated as follows: A hypothesis H explains E^+ and E^- through the BK , if

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

Essentially, the hypothesis, together with the background knowledge, entails all the positive examples and none of the negative examples. In addition to the learning paradigm, one must 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. Many approaches exploit *subsumption* (\leq_{sub}), which has the following property in relation to entailment: $H_1 \leq_{sub} H_2 \Rightarrow H_1 \models H_2$ where H_1 and H_2 are plausible hypotheses. Subsumption provides a measure of specificity between hypotheses which is used, in turn, to measure progress. The FOIL [22] approach to inductive synthesis (*top-down*) iteratively builds logic programs using this principle. *Bottom-up* approaches, i.e. Progol [16], build the subsumptively most specific clause for each positive example and use FOIL to extend more general clauses towards it.

The ILP system *Metagol* [5] was the first system developed implementing the meta-learning approach to search. It uses 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 *template* representation used by δ ILP and our work.

Differentiable ILP

Given that δ ILP plays an integral role in our work, we go into some detail concerning the system architecture. However, due to space constraints, we cannot cover all aspects of the work, and thus we refer to reader to [11] for more details.

With δ ILP, *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 component of a template denotes a clause definition with at most two literals and with at most a defined number of *existential*¹ variables per clause (typically ≤ 1). Additionally, each component of a template is associated with a flag denoting whether it can contain *intensional* definitions; that is, predicates defined by a template and thus not occurring in the BK . This flag also activates recursion, i.e. self-referential definitions.

A *template* is a pair of components where at least one position in the pair is non-empty. The template $((0, false), (1, true))$ accepts (among other clause pairs):

$$p(x, y) : -succ(x, y) \quad p(x, y) : -succ(x, z), p(x, z)$$

The $(0, false)$ component matches the left clause of the pair as it contains 0 existential variables and does not refer to an *intensional* definition (*succ* is in the BK). The $(1, true)$ component matches the right clause as it contains 1 existential variable, namely z , and calls itself, i.e. $p(x, z)$ occurs in its body. This clause structure is allowed as the *intensional* flag is set to *true*. While this definition structure can encode higher arity predicate definitions, learning such predicates is theoretically challenging for this ILP setting [17], and thus restricting oneself to dyadic predicates is a common practice. This restriction is especially important for our work as we will introduce many templates (up to 150).

Evans and Grefenstette designed the *template* structure so the user can precisely define the solution structure, thus simplifying the search. In this work, we use the simplified template $((1, true), (1, true))$ for all *intensional* predicates, i.e. the most general and useful template structure definable using the construction presented in [11]. This design choice results in a larger hypothesis space and, thus, a more challenging experimental setting than what was considered in [11].

δ ILP takes as input a set of templates (denoted p_1, \dots, p_n) and the BK and derives a satisfiability problem where each disjunctive clause $C_{i,j}$ denotes the range of possible choices for clause j given template p_i . The logical models satisfying this formula denote logic programs modulo the clauses derivable using the template instantiated by the BK . Switching from a discrete semantics over $\{0, 1\}$ to a continuous semantics allows δ ILP to exploit the semantics of differentiable logical operators when constructing models and implementing differentiable deduction. Solving ILP tasks, in this setting, is reduced to minimizing loss through gradient descent.

δ ILP uses the input examples E^+ and E^- 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 templates, and L the symbolic language used to describe the problem containing a finite set of atoms.

Each template $p_i = (t_1^i, t_2^i)$ 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 . Given this construction, one can compute a rough approximation (*quartic*) of the number of weights in terms of the number of templates (as each definition requires 2 clauses and each clause requires 2 body predicates). Each weight denotes δ ILP's confidence in a given pair of clauses correctly defining the template p_i . The Authors referred to this construction of the weight tensor as *splitting per definition*. We provide a detailed discussion of *splitting* in the **Contributions**.

δ ILP implements differentiable inferencing by providing each clause c with a function $f_c : [0, 1]^m \rightarrow [0, 1]^m$ whose domain and range are valuations of grounded templates. Note, m is not the number of templates; rather, it is the number of groundings of each template, a much larger number dependent on the BK , language bias, and the *atoms* contained in the symbolic language L . Consider a template p_1 admitting the clause pair (c_1, c_2) , and let the current valuation be \mathcal{EV}_i and $g : [0, 1] \times [0, 1] \rightarrow [0, 1]$ 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)

¹ Not present in the head of the clause.

over all 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 sums, in addition to \mathcal{EV}_i , results in \mathcal{EV}_{i+1} . This process is repeated n times (the desired number of forward-chaining steps), where \mathcal{EV}_0 is derived from the *BK*.

The above construction still depends on a precise definition of f_c . Let $c_g = p(x_1, x_2) :- Q_1(y_1, y_2), Q_2(y_3, y_4)$ 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 complete computation of the tensor required for a step of *forward-chaining*.

Four differentiable operations parameterize the above process for conjunction and disjunction. We will leave further discussion of these operators to **Methodology**.

Related Approaches

To the best of our knowledge, three recent investigations are related to δ ILP and build on the architecture. The *Logical Neural Network* (LNN) based ILP system presented in [24] use similar templating, but only to learn non-recursive *chain rules*, i.e. of the form $p_0(X, Y) :- p_1(X, Z_1), \dots, p_n(Z_n, Y)$. Note that this is easily simulated using δ ILP templates, especially for learning the short rules presented by the authors. Their investigation focused on particular parameterized, differentiable, logical operators optimizable for ILP.

Another system motivated by δ ILP is the α ilp [25] system. The authors focused on learning logic programs that recognize visual scenes rather than general ILP tasks. Instead of templating, the authors use a restricted *BK* that contains predicates to explain the visual scenes used for experimental evaluation. To build clauses, the authors start from a set of initial clauses and use a *top-k beam search* to iteratively extend the body of the clauses based on how they evaluate with respect to the positive examples. In this setting, predicate invention and recursive definitions are not considered. Additionally, the structure of initial clauses is simpler as tasks requiring learning a relation between two variables are not considered.

Feed-Forward Neural-Symbolic Learner [6] does not directly build on the δ ILP architecture but provides an alternative approach to one of the problems the δ ILP investigation addressed, namely developing a neural-symbolic architecture that can provide symbolic rules when presented with noisy input. In this work [6], rather than softening implication and working with fuzzy versions of logical operators as was done by the δ ILP investigation, the authors compose the Inductive *answer set programming* learners *ILASP* [13] and *FAST-LAS* [12] with various pre-trained neural architectures for classification tasks. This data is then transformed into weighted knowledge for the symbolic learner. While this work is to some extent relevant to the investigation outlined in this paper, we focus on improving the differentiable implication mechanism developed by the authors of δ ILP rather than completely replacing it. Furthermore, the authors

focus on tasks with simpler logical structure, similar to the approach taken by α ilp.

Earlier investigations, such as *NeuraILP* [30], experimented with various *T-norms* as differentiable operators and part of the investigation reported in [11] studied their influence on learning. The authors leave scaling their approach to larger, possibly recursive programs as future work, a limitation addressed herein.

In [26], the authors built upon δ ILP but further restricted 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* [9], 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 δ ILP.

Some related systems loosely related to our work are *Logical Tensor Networks* [8], *Lifted Relational Neural Networks* [27], *Neural theorem prover* [15], and *DeepProbLog* [14]. 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. Concerning the explainability aspects of systems similar to δ ILP, one notable mention is *Logic Explained Networks* [2], which adapts the input format of a neural learner such that an explanation can be derived from the output. Though, the problem they tackle is only loosely connected to our work.

Contributions

As mentioned above in subsection *Differentiable ILP*, the number of weights used by δ ILP is approximately *quartic* in the number of templates used, thus incurring a significant memory footprint. This issue is further exacerbated by differentiable inferencing as a grounding of the hypothesis space is needed to compute evaluations. As a result, experiments performed in [11] use task-specific templates precisely defining the structure of the solution; this is evident in their experiments where for certain tasks, 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. This additional *language bias* greatly influences the success rate of δ ILP on tasks such as *length of a list* upon which the authors report low loss in **92.5%** of all runs. These results significantly differ from our reduced *language bias* experiments resulting in **0%** of solutions correctly classifying training and test data and **0%** achieving low loss.

The necessity of these restrictions partially entails the Author's choice to split programs *per template* and assign weights to each instance; their design choice results in a large vector v of learnable parameters. While this seems to imply high dimensionality, as discussed above, *softmax* is applied to v during differentiable inferencing and thus transforms v into a distribution, effectively reducing its dimensionality in the process.

Our investigation aims to: (i) increase the dimensionality of the search space while maintaining the efficacy of the differentiable inferencing implemented in δ ILP, and (ii) minimize the *bias* required for effective learning. We proceed by adding many auxiliary predicate definitions with uniformly defined templates as discussed in subsection *Differentiable Inferencing*, i.e. templates of the form

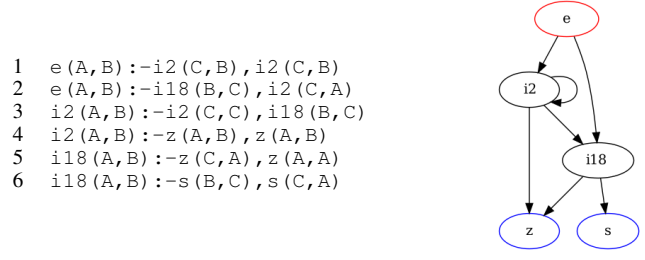
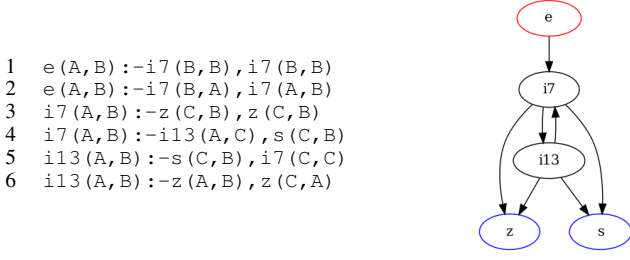


Figure 1: *even* (*e* above) solutions using 20 predicate definitions, trimmed to used definitions. Note, *s* denotes successor, and *z* denotes zero.

((1,true),(1,true)); this largely reduces the biases towards solutions of a particular shape. Nonetheless, given the significant number of weights δ ILP requires for learning, large-scale predicate invention is highly intractable. Thus, we amend how weights are assigned to templates to implement our approach. The design choice made in [11] to assign weights to each possible pair of clauses, thus splitting programs by templates (*per template* splitting), is the main source of its significant memory footprint. The authors discuss this design choice in Appendix F of their paper [11]. While the authors attempted to split *per clause*, without large-scale predicate invention, they found that this approach was “incapable of escaping local minima on harder tasks”; splitting *per clause* rather than splitting *per template* results in a quadratic reduction in the number of assigned weights.

In our work, we go one step further and split *per literal*, resulting in another quadratic reduction in the number of assigned weights. An illustration of the different splits can be found in Figure 2. One can read the weights assigned when splitting *per literal* as the likelihood that a given predicate *p* will occur in one of the two clauses of a template *t*. We refer to this new system as δ ILP₂.

Additionally, we observed that the most challenging tasks require learning a binary relation whose solution requires using a third (non-argument) variable, an additional existential variable. This observation differs from the observations presented in [11] where complexity was measured purely in terms of the number of intensional predicates required to solve the task. For example, consider *buzz* (*w*. only +1), which our approach solves 61% of the time, and $Y=X+4$, which our approach solves 4% of the time. Note that *buzz* requires four intensional predicates while $Y=X+4$ only requires two, yet unlike *buzz*, both of those predicates relate two variables through a third non-argument existential variable.

We test δ ILP₂ and support our observation through experimentally testing the three hypotheses listed below (see **Experiments**):

- **Hypothesis 1:** Differentiable inductive logic programming benefits from increasing the number of intensional predicates used during training.
- **Hypothesis 2:** The benefit suggested by *hypothesis 1* is not solely due to the relationship between increasing the number of intensional predicates and training a multitude of times with a task-specific number of intensional predicates.
- **Hypothesis 3:** Tasks involving learning a binary predicate that uses existential quantification over an additional variable remain a challenge regardless of the approach to splitting taken.

Methodology

In this section, we outline the methodological differences in comparison to standard implementations of differentiable inferencing;

namely, we (i) split the program by body predicates, (ii) use slightly different differentiable logical operators, (iii) use more precise measures of training outcomes, and (iv) use a slightly different method of batching examples. We cover these differences below.

Modifications to original δ ILP

Splitting the program

The focus of the work presented in this paper is to highlight the benefits of large-scale predicate invention when training inductive synthesizers based on gradient descent. However, as mentioned earlier, certain features of δ ILP entail a large memory footprint and thus make large-scale predicate invention during training intractable. We adjust the weight assignment to templates to alleviate this problem, i.e., splitting predicate definitions. As discussed in the previous section, we implemented *per Literal* splitting. As *per Literal* splitting significantly reduces the memory footprint and, to some extent, simplifies computation as we can include a plethora of templates during training without exhausting resources.

T-norm for fuzzy logic

As discussed in **Related Work**, δ ILP 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 δ ILP experimented with *t-norms* or (*triangle norms*), continuous versions of *classical* conjunction [10] from which one can derive continuous versions of other logical operators. 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.

	δ ILP	δ ILP ₂
\wedge -Literal	product	product
\vee -Exists	max	max
\vee -Clausal	max	max
\vee -Step	product	max

When computing many inference steps, *product* produces vanishingly small gradients. We require computing more inference steps as we are producing much larger programs (See Figure 7 & 4). Thus, we use *max* for \vee -step.

$[gp(A, B) :- il(A, C'), il(C, B)]$	$[gp(A, B) :- il(A, C), il(C, B)]$	$gp(A, B) :- [il(A, C)], [il(C, B)]$
$[il(A, B) :- mom(A, B), mom(A, B)]$	$[il(A, B) :- mom(A, B), mom(A, B)]$	$il(A, B) :- [mom(A, B)], [mom(A, B)]$
$[il(A, B) :- dad(A, B), dad(A, B)]$	$[il(A, B) :- dad(A, B), dad(A, B)]$	$il(A, B) :- [dad(A, B)], [dad(A, B)]$

Figure 2: Splits of the “grandparent” predicate definition – per template, clauses, and literals (denoted by $[]$)

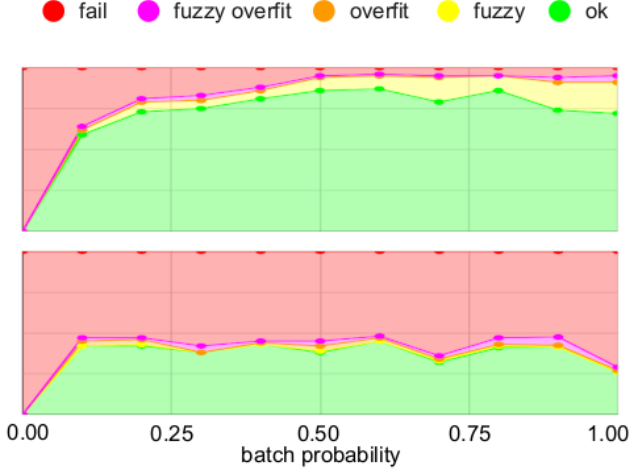


Figure 3: Batch Probability: *fizz* (Top) and *member* (Bottom)

Batch probability

Our approach requires computing values for all predicates over all combinations of atoms, thus motivating an alternative approach to the typical implementations of mini-batching. Instead of parameterization by *batch size*, we use a *batch probability* – the likelihood of an example contributing to gradient computation.

When computing the loss, the example sets E^+ and E^- equally contribute. Regardless of the chosen examples, the loss is balanced (divided by the number of examples contributing). If batching results in no examples from E^+ (E^-), we set that half of the loss to 0 (with 0 gradient).

Figure 3 illustrates the influence of *batch probability* on performance, which degrades only in the vicinity of 0.0 and 1.0. In all other experiments, we used the default value of 0.5.

Considered outcomes

Overfitting

The experimental design presented by the authors of δ ILP avoids overfitting as the search space is restricted enough to exclude programs that fit the training data and are not correct solutions. However, this method of avoiding overfitting is no longer viable when 100s of templates are used during training as it is much easier for the synthesizer to encode the examples.

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

During experimentation, we observed that δ ILP₂ rarely overfits, even when it clearly could. A plausible explanation is that shorter, precise solutions have a higher frequency in the search space.

Fuzzy solutions

Another class of solutions – which affect both δ ILP and our approach – is the class of fuzzy solutions. We define this type of solution as follows: when training results in a model that correctly predicts the answer using fuzzy logic, but the program is incorrect when collapsed to a classical logic program (using the predicates with the highest weight). 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. Entirely correct solutions for *even* are translatable into a program correct for all natural numbers, while a fuzzy solution fails to generalize beyond what it has seen during training.

Experiments

task	Binary Rel.	correct on test	fuzzily correct on test	correct on training	fuzzily correct on training	change correct on test	change correct on training	change signif.
predecessor	✓	100	100	100	100	+2%	+2%	- -
even		92	99	92	99	+32%	+32%	✓ ✓
$(X \leq Y)^*$	✓	30	31	35	38	-70%	-65%	✓ ✓
fizz		91	97	91	97	+91%	+91%	✓ ✓
buzz w. +2 & +3		77	80	97	100	+77%	+97%	✓ ✓
buzz w. only +1		61	65	61	65	+61%	+65%	✓ ✓
$Y=X+2$	✓	99	100	99	100	-1%	-1%	- -
$Y=X+4$	✓	4	12	5	13	+4%	+5%	- -
member*	✓	17	19	37	43	-67%	-67%	✓ ✓
length	✓	25	26	31	38	+25%	+31%	✓ ✓
grandparent	✓	38	38	92	94	+38%	+89%	✓ ✓
undirected edge	✓	94	94	100	100	+75%	+81%	✓ ✓
adjacent to red		94	99	94	99	+48%	+48%	✓ ✓
two children		74	100	74	100	+13%	+13%	✓ ✓
graph colouring		83	85	96	100	-15%	-2%	✓ -
connectedness	✓	40	41	98	99	+16%	+74%	✓ ✓
cyclic		19	19	90	100	+18%	+89%	✓ ✓

Table 1: *Split Per Literal* results: Change computed with respect to Table 2. Significance computed using t-test and $p < 1e^{-4}$. Problems marked with a * are significantly easier for *split by template* as the entire hypothesis space fits in one weight matrix.

We compare δ ILP₂ (using *per Literal* splitting) to δ ILP (using *per Template* splitting) on tasks presented in [11] plus additional tasks to experimentally test *Hypothesis 2 & 3*. The results are shown in Table 1 & 2. Concerning experimental parameters, we ran δ ILP₂ using 150 invented predicates to produce the results in Table 1. For δ ILP, we ran it with the precise number of invented predicates needed to solve the task. Using more invented predicates was infeasible for many tasks due to the large memory footprint of splitting *per template*. In both cases, we associated all invented predicates with the generic templates $((1, \text{true}), (1, \text{true}))$. Other parameters are as follows:

- running for $2k$ gradient descent steps, ending early when loss reaches 10^{-3}
- differentiable inference is performed for 25 steps
- batch probability of 0.5 (each example has 0.5 chance is being part of any given batch)
- weights are initialized using a normal distribution
- output programs are derived by selecting the highest weighted literals for each template.

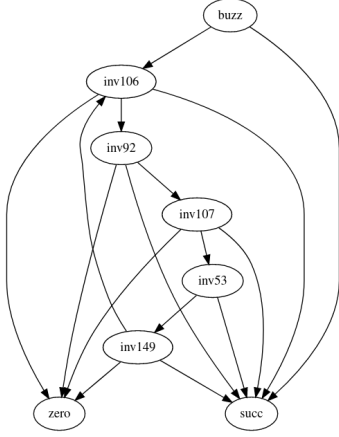


Figure 4: Simply dependency graph of a program learned by δILP_2 training on the *buzz* w. +1 task. This program correctly generalized to the test set.

We ran the experiments producing Figure 3 & 5 on a computational cluster with 16 nodes, each with 4 GeForce RTX 2070 (with 8 GB of RAM) GPUs. We ran the experiments producing Table 1 & 2 on a GPU server with 8 NVIDIA A40 GPUs (46GB each). We implemented both δILP and δILP_2 using PyTorch [20] (version 2.0).

As defined in [11], some tasks use excessively sparse training data. This choice is sufficient for δILP as overfitting is not possible. 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.

Hypothesis 1

task	Binary Rel.	all correct	fuzzily correct	correct on training	fuzzily correct on training
predecessor	✓	98	100	98	100
even		70	94	70	94
$(X \leq Y)^*$	✓	100	100	100	100
fizz		0	0	0	0
<i>buzz</i> w. +2 & +3		-	-	-	-
<i>buzz</i> w. only +1		-	-	-	-
$Y=X+2$	✓	100	100	100	100
$Y=X+4$	✓	0	0	0	0
member*	✓	84	100	84	100
length	✓	0	0	0	0
grandparent	✓	0	0	3	3
undirected edge	✓	19	100	19	100
adjacent to red		46	90	46	90
two children		61	100	61	100
graph colouring		98	100	98	100
connectedness	✓	24	100	24	100
cyclic		0	0	1	1

Table 2: Split Per Template Result: Due to significant memory requirements, neither *buzz* tasks fit in GPU memory (roughly 46GB). We extrapolate results for these tasks from *fizz*.

Strong evidence for this hypothesis can be seen in Figures 5 and 6, clearly showing that using much more intensional predicates than necessary is beneficial.

When comparing with δILP , out of the 17 tasks we tested δILP and δILP_2 on, δILP_2 showed improved performance on 13 tasks, and the improved performance was statistically significant for 12 of these tasks. Of the four remaining tasks, δILP showed a statistically significant performance difference on two, namely $X \leq Y$ and *member*.

The latter benefits from splitting per template as the entire search space fits into one weight matrix. Thus, δILP is essentially performing brute force search. While one would expect the same issue to occur for *connectedness*, there are fewer solutions to *member* and $X \leq Y$ in the search space than in the case of *connectedness*; it is a more general concept. Thus, even when δILP has an advantage, δILP_2 outperforms it when training on more complex learning tasks.

Notably, δILP_2 outperforms δILP on many challenging tasks. For example, *buzz* cannot be solved by δILP without specifying the solution shape. Even then, it attained low loss only 14% of the time when allowing +2 and +3 in the *BK* [11]. In contrast, we achieved a 61% success rate on this task even when *BK* contained only *successor* and *zero*; for the dependency graphs of solutions learned, see Figure 4 & 7.

Hypothesis 2

task	all correct	fuzzily correct	correct on training	fuzzily correct on training
3 invented predicates	0%	0.02%	0%	0.02%
5 invented predicates	0.11%	0.13%	0.11%	0.14%
50 invented predicates	60%	65%	60%	65%

Table 3: Results of running δILP_2 on *divisible by 6* with a varying number of predicates. For 3 and 5 invented predicates we ran δILP_2 10k times. For 50 invented predicates, we ran δILP_2 100 times.

To illustrate that large-scale predicate invention is not equivalent to re-initialization of weights, we ran δILP_2 on the *divisible by 6* task while varying the numbers of invented predicates used during training (results shown in Table 3). Note, *divisible by 6* is a slightly more complex task than *buzz* and thus aids in illustrating the effect of large-scale predicate invention.

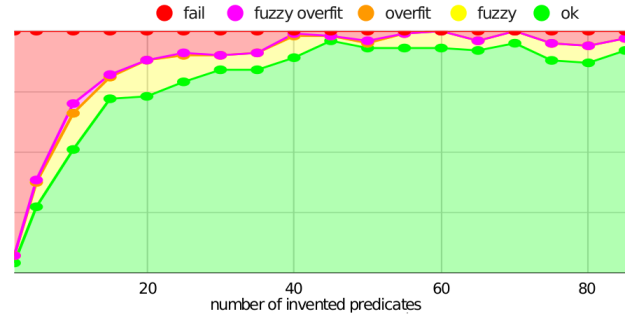


Figure 5: Results of changing the number of invented predicates in *fizz*. The number of intensional predicates ranged between 1 and 85.

According to results in table 3, when training with three invented predicates (minimum required), we would need to run δILP_2 5000 times to achieve a similar probability of success (finding a fuzzily correct solution) as a single 50 predicate run.

When using five invented predicates, which is minimum required to avoid constructing binary predicates (see Hypothesis 3), we would need to run δILP_2 750 times to achieve a similar probability of success (finding a fuzzily correct solution) as a single 50 predicate run.

These results do not necessarily imply that using more predicates is always beneficial over doing multiple runs. However, the above results show that repeated training with intermediary weight re-initialization is not a sufficient explanation of the observed benefits of large-scale predicate invention during training.

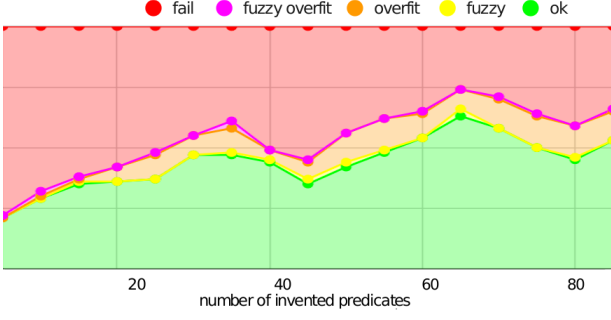


Figure 6: Results of changing the number of invented predicates in $X \leq Y$. The number of intensional predicates ranged between 1 and 85.

Hypothesis 3

In Table 1 & 2, one can observe that some tasks requiring the system to learn a relatively simple program (i.e. *length*) are more challenging than tasks such as *buzz* that require learning a much larger program.

As stated above, we hypothesize this is due to propagating the gradient through an existential quantification on the third variable. This results in difficulties when learning programs for tasks that require a binary predicate. The difficulty increases with the number of binary predicate predicates with this property required to solve the tasks.

We introduced an additional task explicitly designed to test this hypothesis: the *plus 4* predicate ($Y = X + 4$). This task requires only one more predicate than *plus 2*, yet the success rate drops significantly with respect to *plus 2* (from 99% to 4%). For *divisibility by 2* (*even*) and *derivability by 5* (*buzz*), the change is not as steep (from 92% to 61%).

Thus, the number of relational predicate definitions that a given task requires the system to learn is a more precise measure of complexity than the number of intensional predicates the system must learn.

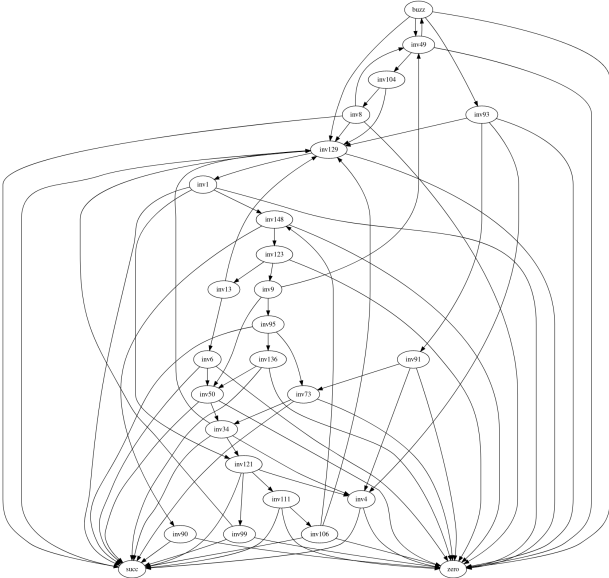


Figure 7: Complex dependency graphs of a program learned by δILP_2 training on the *buzz* w. +1 task. This program correctly generalized to the test set.

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 . We performed our experiments using reduced language bias compared to the experiments presented in [11]. Furthermore, we used the same generic template for all predicate definitions learned by the system. This choice makes some tasks significantly more difficult. Additionally, we verified that the performance gains were not simply due to properties shared with weight re-initialization when using a task-specific number of invented predicates during the learning process (*Hypothesis 2*).

During experimentation, we noticed that the difficulty of the task did not correlate well with the number of intensional predicates needed to solve it but rather with the arity and the necessity of a third existential variable. Therefore, we tested this conjecture using the tasks $Y = X + 2$ and $Y = X + 4$. While both systems solve $Y = X + 2$, performance drastically drops for $Y = X + 4$, which only requires learning two invented predicates. Note *buzz* requires learning four invented predicates and is easily solved by δILP_2 . This observation highlights the challenging tasks for such inductive synthesis approaches (*Hypothesis 3*) and suggests a direction for future investigation.

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 Authors of δILP presented the first steps in [11]. One can imagine the development of a network inferring a discrete set of objects in an image [1], or integration with Transformer-based [28] language models that produce atoms δILP_2 can process. This research direction 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 [29], 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 solve more complex tasks. A similar idea would be decomposing a program into several small parts and learning by gradient descent using a different programming paradigm, such as functional programming.

Finally, we can improve our approach 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 toward 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. We would also like to Thank David Coufal (CAS ICS) for setting up and providing access to the institute’s GPU Server.

References

- [1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko, ‘End-to-end object detection with transformers’, *CoRR*, **abs/2005.12872**, (2020).
- [2] Gabriele Ciravegna, Pietro Barbiero, Francesco Giannini, Marco Gori, Pietro Liò, Marco Maggini, and Stefano Melacci, ‘Logic explained networks’, *Artif. Intell.*, **314**, 103822, (2023).

- [3] Andrew Cropper and Sebastijan Dumancic, 'Inductive logic programming at 30: A new introduction', *J. Artif. Intell. Res.*, **74**, 765–850, (2022).
- [4] Andrew Cropper and Rolf Morel, 'Learning programs by learning from failures', *Mach. Learn.*, **110**(4), 801–856, (2021).
- [5] Andrew Cropper and Stephen Muggleton. Metagol system. github.com/metagol/metagol, 2016.
- [6] Daniel Cunningham, Mark Law, Jorge Lobo, and Alessandra Russo, 'FFNSL: feed-forward neural-symbolic learner', *Mach. Learn.*, **112**(2), 515–569, (2023).
- [7] Wang-Zhou Dai, Stephen H. Muggleton, Jing Wen, Alireza Tamaddoni-Nezhad, and Zhi-Hua Zhou, 'Logical vision: One-shot meta-interpretive learning from real images', in *ILP 2017*, eds., Nicolas Lachiche and Christel Vrain, volume 10759 of *LNCs*, pp. 46–62. Springer, (2017).
- [8] Ivan Donadello, Luciano Serafini, and Artur S. d'Avila Garcez, 'Logic tensor networks for semantic image interpretation', in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, ed., Carles Sierra, pp. 1596–1602. [ijcai.org](https://www.ijcai.org), (2017).
- [9] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou, 'Neural logic machines', in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, (2019).
- [10] Francesc Esteva and LLuis Godo, 'Monoidal t-norm based logic: towards a logic for left-continuous t-norms', *Fuzzy Sets and Systems*, **124**(3), 271–288, (2001). Fuzzy Logic.
- [11] Richard Evans and Edward Grefenstette, 'Learning explanatory rules from noisy data', *Journal of Artificial Intelligence Research*, **61**, 1–64, (2018).
- [12] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo, 'Fastlas: Scalable inductive logic programming incorporating domain-specific optimisation criteria', 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*, pp. 2877–2885. AAAI Press, (2020).
- [13] Mark Law, Alessandra Russo, and Krysia Broda, 'Inductive learning of answer set programs', in *Proceedings of Logics in Artificial Intelligence*, pp. 311–325. Springer, (August 2014).
- [14] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt, 'Neural probabilistic logic programming in deepproblog', *Artif. Intell.*, **298**, 103504, (2021).
- [15] Pasquale Minervini, Matko Bosnjak, Tim Rocktäschel, Sebastian Riedel, and Edward Grefenstette, '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*, pp. 5182–5190. AAAI Press, (2020).
- [16] Stephen Muggleton, 'Inverse entailment and progol', *New Generation Computing*, **13**(3&4), 245–286, (December 1995).
- [17] Stephen H. Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad, 'Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited', *Mach. Learn.*, **100**(1), 49–73, (2015).
- [18] Stephen H. Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter A. Flach, Katsumi Inoue, and Ashwin Srinivasan, 'ILP turns 20 - biography and future challenges', *Mach. Learn.*, **86**(1), 3–23, (2012).
- [19] S. H. Muggleton FREng. Supplementary material from "hypothesizing an algorithm from one example: the role of specificity", Apr 2023.
- [20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, 'Pytorch: An imperative style, high-performance deep learning library', in *Advances in Neural Information Processing Systems 32*, eds., H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, 8024–8035, Curran Associates, Inc., (2019).
- [21] Stanislaw J. Purgal, David M. Cerna, and Cezary Kaliszzyk, 'Learning higher-order logic programs from failures', in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, ed., Luc De Raedt, pp. 2726–2733. [ijcai.org](https://www.ijcai.org), (2022).
- [22] J. Ross Quinlan, 'Learning logical definitions from relations', *Mach. Learn.*, **5**, 239–266, (1990).
- [23] Luc De Raedt, *Logical and relational learning*, Cognitive Technologies, Springer, 2008.
- [24] Prithviraj Sen, Breno W. S. R. de Carvalho, Ryan Riegel, and Alexander G. Gray, '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*, pp. 8212–8219. AAAI Press, (2022).
- [25] H. shindo, V. Pfanschilling, and D.S. Dhami, 'αilp: thinking visual scenes as differentiable logic programs', *Machine Learning*, (2023).
- [26] Hikaru Shindo, Masaaki Nishino, and Akihiro Yamamoto, '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*, pp. 5034–5041. AAAI Press, (2021).
- [27] Gustav Sourek, Martin Svatos, Filip Zelezny, Steven Schockaert, and Ondrej Kuzelka, 'Stacked structure learning for lifted relational neural networks', in *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, September 4-6, 2017, Revised Selected Papers*, eds., Nicolas Lachiche and Christel Vrain, volume 10759 of *Lecture Notes in Computer Science*, pp. 140–151. Springer, (2017).
- [28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, 'Attention is all you need', *CoRR*, **abs/1706.03762**, (2017).
- [29] Ronald J. Williams, 'Simple statistical gradient-following algorithms for connectionist reinforcement learning', *Mach. Learn.*, **8**, 229–256, (1992).
- [30] Fan Yang, Zhilin Yang, and William W. Cohen, 'Differentiable learning of logical rules for knowledge base reasoning', in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, eds., Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, pp. 2319–2328, (2017).