

Neural-Symbolic Predicate Invention: Learning Relational Concepts from Visual Scenes

Dmitry S. Kulyabov^{1,2,*†}, Ilaria Tiddi^{3†} and Manfred Jeusfeld^{4†}

¹Peoples' Friendship University of Russia (RUDN University), 6 Miklukho-Maklaya St, Moscow, 117198, Russian Federation

²Joint Institute for Nuclear Research, 6 Joliot-Curie, Dubna, Moscow region, 141980, Russian Federation

³Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands

⁴University of Skövde, Högskovvägen 1, 541 28 Skövde, Sweden

Abstract

The predicates used for Inductive Logic Programming (ILP) systems are typically elusive and need to be hand-crafted in advance limiting the generalization of the system when learning new rules without sufficient background knowledge. Predicate Invention (PI) for ILP is the problem of discovering new concepts that describe hidden relationships in the domain. PI can mitigate the generalization problem by inferring new concepts such that the system gains better vocabularies to compose logic rules to solve problems. Although several PI approaches for symbolic ILP systems exist, PI for NeSy ILP systems, which can deal with visual inputs to learn logic rules using differentiable reasoning, is relatively unaddressed. To this end, we propose a neural-symbolic approach, NeSy- π , to invent predicates from visual scenes for NeSy ILP systems based on clustering and extension of relational concepts. NeSy- π handles visual scenes as its input using deep neural networks for the visual perception, and invents new concepts which are useful to solve the task of classifying complex visual scenes. The invented concepts can be used by any NeSy ILP systems instead of hand-crafted background knowledge. Our experiments show that the PI model is capable of inventing high-level concepts and solving complex visual logical patterns more efficiently in absence of explicit background knowledge. Moreover, the invented concepts are explainable and interpretable while also providing competitive results with the state of the art NeSy ILP systems with given knowledge.

Keywords

Predicate Invention, Inductive Logic Programming, Neural Symbolic Artificial Intelligence

1. Introduction

Inductive Logic Programming (ILP) is a set of techniques to learn generalized logic programs given data [? ? ?]. In contrast to Deep Neural Networks (DNNs), ILP gains vital advantages, e.g. it can learn explanatory rules from small data. However, predicates for ILP systems are typically elusive and need to be hand-crafted and thus they require much priors to compose

Woodstock'22: Symposium on the irreproducible science, June 07–11, 2022, Woodstock, NY

*Corresponding author.

†These authors contributed equally.


✉ kulyabov-ds@rudn.ru (D. S. Kulyabov); i.tiddi@vu.nl (I. Tiddi); Manfred.Jeusfeld@acm.org (M. Jeusfeld)

🌐 <https://yamadharma.github.io/> (D. S. Kulyabov); <https://kmitd.github.io/ilaria/> (I. Tiddi);

<http://conceptbase.sourceforge.net/mjf/> (M. Jeusfeld)

🆔 0000-0002-0877-7063 (D. S. Kulyabov); 0000-0001-7116-9338 (I. Tiddi); 0000-0002-9421-8566 (M. Jeusfeld)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

	NeSy-ILP	NeSy- π
Const	Object : 01/02, Shape : Sphere/Cube, Color : Blue/Pink/Green , Direction : Left/Right/Front/Behind	
Basic-Pred	color/2/obj, color; shape/2/obj, shape; dir/2/obj, obj	
BK	b_sp(01) : \neg color(01, B), shape(01, Sp). g_sp(01) : \neg color(01, G), shape(01, Sp). g_cu(01) : \neg color(01, G), shape(01, Cu).	-
Pred	left_side/2/obj, obj; right_side/2/obj, obj	-

Figure 1: A logic Pattern in 3D scenes can be learned by Neural-Symbolic ILP system. **Left:** Positive images on the first row, negative images on the second row. The truth pattern is: a blue sphere either locate on the left side of a green sphere or locate on the right side of a green cube. **Right:** Comparison of language requirement between NeSy-ILP and NeSy- π . Meaning of abbreviations in the table: Const-Constant. BK-Background Knowledge. Ne-Pred-Neural Predicate. B-Blue, G-Green, Sp-Sphere, Cu-Cube.

solutions. Predicate invention (PI) systems invent new predicates as symbols for new concepts from well designed primitive predicates, which enlarge the expression of the language in ILP and consequently reduce the dependence on human experts [?]. One simple example is the concept of a sphere with blue color, in NeSy-ILP system, the concept `blue_sphere` can be explained by a clause `blue_sphere(X) : \neg Color(X, blue), Shape(X, sphere)`. which is given as BK knowledge, but with PI system, such concepts are learned from separate basic predicates `Color(X, blue)` and `Shape(X, sphere)` by concatenation.

Recently, neural-symbolic ILP frameworks have been proposed [8?], which embrace DNNs for visual perception and learn explanatory rules from raw inputs. NeSy-ILP systems overcome pure neural-based baselines on visual reasoning tasks where the answers are derived by reasoning about attributes of the objects and their relations. However, current NeSy-ILP systems has a crucial drawback of assuming all of the predicates to be needed are given as background knowledge., *e.g.* pretrained neural predicates or explained by hand-crafted background knowledge, and thus optimal programs are hard to be discovered without sufficient background knowledge. Moreover, the collection of background knowledge is very costly, since it needs to be provided by human experts or requires pre-training of neural modules with additional supervision. This limits severely the applicability of the NeSy-ILP systems to different domains. In contrast, DNNs such as Transformers require a minimal prior and gain high performance by learning from data, and thus have been successfully applied to various different domains. How can we realize a NeSy-ILP system that can learn from less background knowledge? **[highlight spatial relationships as the issue -HK]**

To mitigate this issue, we propose Neural-Symbolic Predicate Invention (NeSy- π), a predicate invention pipeline that can handle visual scenes. NeSy- π invents relational concepts given visual scenes by finding rules defining them using primitive predicates. NeSy- π can be integrated with existing NeSy-ILP systems so that they gain rich vocabulary to compose solutions efficiently. NeSy- π reduces the amount of priors to be given by human experts or pre-training of neural

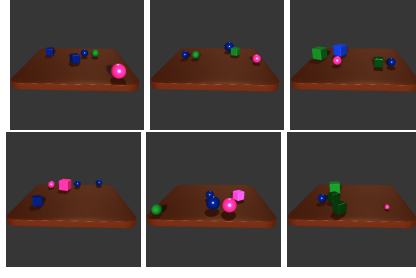


Figure 2: A logic Pattern in 3D scenes can be learned by Neural-Symbolic ILP system. Positive images on the first row, negative images on the second row. The truth pattern is: a blue sphere either locate on the left side of a green sphere or locate on the right side of a green cube.

Table 1

Comparison of language requirement between NeSy-ILP and NeSy- π . Meaning of abbreviations in the table: Const-Constant. BK-Background Knowledge. Ne-Pred-Neural Predicate. [where to put this table? -JING] [besides Fig 1 as before -DEV]

	NeSy-ILP	NeSy- π
Const	Object : 01/02, Color : Blue/Pink/Green Shape : Sphere/Cube , Direction : Left/Right/Front/Behind	
Basic-Pred	color/2/obj, color; shape/2/obj, shape; dir/2/obj, obj	
BK	b_sp(01) : \neg color(01, Blue), shape(01, Sphere). g_sp(01) : \neg color(01, Green), shape(01, Sphere). g_cu(01) : \neg color(01, Green), shape(01, Cube).	-
Ne-Pred	left_side/2/obj, obj; right_side/2/obj, obj	-

modules, and thus extends the applicability of NeSy-ILP systems. NeSy- π discovers predicates describing attributes of objects and their spatial relationships in visual scenes. We propose NeSy- π , a neural-symbolic PI approach, based on *clustering* and *extension* of relational concepts, which is able to reasoning visual scenes without background knowledge. The knowledge can be summarized from scenes. It evaluates the clauses based on its characteristics of necessity and sufficiency. The procedure is repeated iteration by iteration, until the target clauses are found. We tested our approach on both 2D and 3D image patterns. For example, the concepts like left, right, nearby supposed to be invented as new predicates during the training if any of them are needed to represent the target pattern in the positive images. This map both considers the distance and directions of the latent relation objects. We designed an evaluation function based on the characteristics of necessity and sufficiency of the clause, which can scoring the searched clauses and keep the promising ones for further extension.

Overall, we make the following important contributions:

- We propose NeSy- π , which is a NeSy predicate invention framework compatible with NeSy ILP systems. NeSy- π extends NeSy ILP systems by providing the capability to enrich their vocabularies by learning from data.
- We develop 3D Kandinsky Patterns, which are user-friendly 3D image-generation systems according to abstract patterns on attributes and relations of objects. It extends Kandinsky

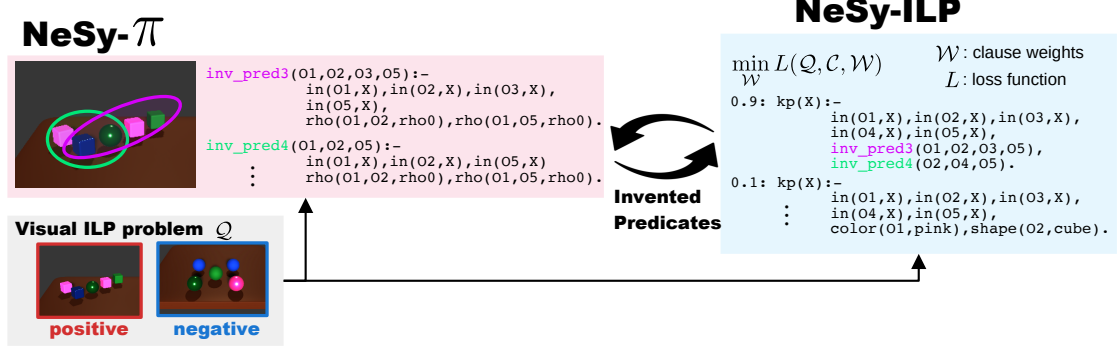


Figure 3: NeSy- π architecture. NeSy- π invents relational concepts given visual scenes. The invented predicates are fed into a NeSy-ILP solver to learn classification rules. NeSy-ILP solver generates candidates of rules using given predicates, performs differentiable reasoning using weighted clauses, and optimize the clause weights by gradient descent. To this end, the classification rules can be composed efficiently using the invented predicates. (Best viewed in color) [first draft, should be improved –HK]
[do we have the opposite arrow: NeSy-ILP \rightarrow NeSy- π in the system? –HK]

patterns to the 3D world, and it achieves faster generation than other environments, e.g. CLEVR [].

- Complex shapes [extend –DEV]
- We discussed about *when* and *how* to invent new predicates based on *necessary* and *sufficient* judgment factors. [rewrite –DEV]

2. First-order Logic and Inductive Logic Programming

First-Order Logic (FOL). A Language \mathcal{L} is a tuple $(\mathcal{P}, \mathcal{A}, \mathcal{F}, \mathcal{V})$, where \mathcal{P} is a set of predicates, \mathcal{A} is a set of constants, \mathcal{F} is a set of function symbols (functors), and \mathcal{V} is a set of variables. A *term* is a constant, a variable, or a term that consists of a functor. A *ground term* is a term with no variables. We denote n -ary predicate p by p/n . An *atom* is a formula $p(t_1, \dots, t_n)$, where p is an n -ary predicate symbol and t_1, \dots, t_n are terms. A *ground atom* or simply a *fact* is an atom with no variables. A *literal* is an atom or its negation. A *positive literal* is just an atom. A *negative literal* is the negation of an atom. A *clause* is a finite disjunction (\vee) of literals. A *ground clause* is a clause with no variables. A *definite clause* is a clause with exactly one positive literal. If A, B_1, \dots, B_n are atoms, then $A \vee \neg B_1 \vee \dots \vee \neg B_n$ is a definite clause. We write definite clauses in the form of $A :- B_1, \dots, B_n$. Atom A is called the *head*, and set of negative atoms $\{B_1, \dots, B_n\}$ is called the *body*. We call definite clauses as clauses for simplicity in this paper. We denote *true* as \top and *false* as \perp . Substitution $\theta = \{X_1 = t_1, \dots, X_n = t_n\}$ is an assignment of term t_i to variable X_i . An application of substitution θ to atom A is written as $A\theta$. An atom is an atomic *formula*. For formula F and G , $\neg F$, $F \wedge G$, and $F \vee G$ are also formulas. *Interpretation* of language \mathcal{L} is a tuple $(\mathcal{D}, \mathcal{I}_A, \mathcal{I}_F, \mathcal{I}_P)$, where \mathcal{D} is the domain, \mathcal{I}_A is the assignments of an element in \mathcal{D} for each constant $a \in \mathcal{A}$, \mathcal{I}_F is the assignments of a function from \mathcal{D}^n to \mathcal{D} for each n -ary function symbol $f \in \mathcal{F}$, and \mathcal{I}_P is the assignments of a function from \mathcal{D}^n to $\{\top, \perp\}$ for each n -ary predicate $p \in \mathcal{P}$. For language \mathcal{L} and formula

X , an interpretation \mathcal{I} is a *model* if the truth value of X w.r.t \mathcal{I} is true. Formula X is a *logical consequence* or *logical entailment* of a set of formulas \mathcal{H} , denoted $\mathcal{H} \models X$, if, \mathcal{I} is a model for \mathcal{H} implies that \mathcal{I} is a model for X for every interpretation \mathcal{I} of \mathcal{L} .

Inductive Logic Programming. ILP problem \mathcal{Q} is tuple $(\mathcal{E}^+, \mathcal{E}^-, \mathcal{B}, \mathcal{L})$, where \mathcal{E}^+ is a set of positive examples, \mathcal{E}^- is a set of negative examples, \mathcal{B} is background knowledge, and \mathcal{L} is a language. We assume that the examples and the background knowledge are ground atoms. The solution to an ILP problem is a set of definite clauses $\mathcal{H} \subseteq \mathcal{L}$ that satisfies the following conditions:

- $\forall A \in \mathcal{E}^+ \mathcal{H} \cup \mathcal{B} \models A.$
- $\forall A \in \mathcal{E}^- \mathcal{H} \cup \mathcal{B} \not\models A.$

Typically the search algorithm starts from general clauses. If the current clauses are too general (strong), i.e., they entail too many negative examples, then the solver incrementally specifies (weakens) them. This weakening operation is called a *refinement*, which is one of the essential tools for ILP.

Ne-Sy Inductive Logic Programming. We address the ILP problem in visual scenes, which is called *visual ILP problem*, where each example is given as an image containing several objects. The classification pattern is defined on high-level concepts such as attributes and relations of objects.

α ILP learns differentiable logic programs that describe complex visual scenes. We basically follow the differentiable ILP setting [8?], where an ILP problem is formulated as an optimization problem that has the following general form:

$$\min_{\mathcal{W}} \text{loss}(\mathcal{Q}, \mathcal{C}, \mathcal{W}), \quad (1)$$

where \mathcal{Q} is an ILP problem, \mathcal{C} is a set of candidates of clauses, \mathcal{W} is a set of weights for clauses, and loss is a loss function that returns a penalty when training constraints are violated. We note that we solve visual ILP problems, where each positive and negative example is an image containing several objects.[\[continue –HK\]](#)

α ILP [?] is a NeSy-ILP system which handles complex visual scenes, which learns to represent scenes as logic programs. Intuitively, logical atoms correspond to objects, attributes, and relations, and clauses encode high-level scene information. α ILP has an end-to-end reasoning architecture from visual inputs and learns differentiable logic programs by gradient descent.

3. Neuro-Symbolic Predicate Invention: NeSy- π

The target of inductive logic programming is to find a target clause P for the positive patterns Q , such that the clause P describes some logical relations that exist and only exist in the positive patterns. Thus the target clause P is sufficient and necessary for the positive patterns Q .

$$P \Leftrightarrow Q$$

Definition 3.1 (PN pair). *A pair of positive and negative images.*

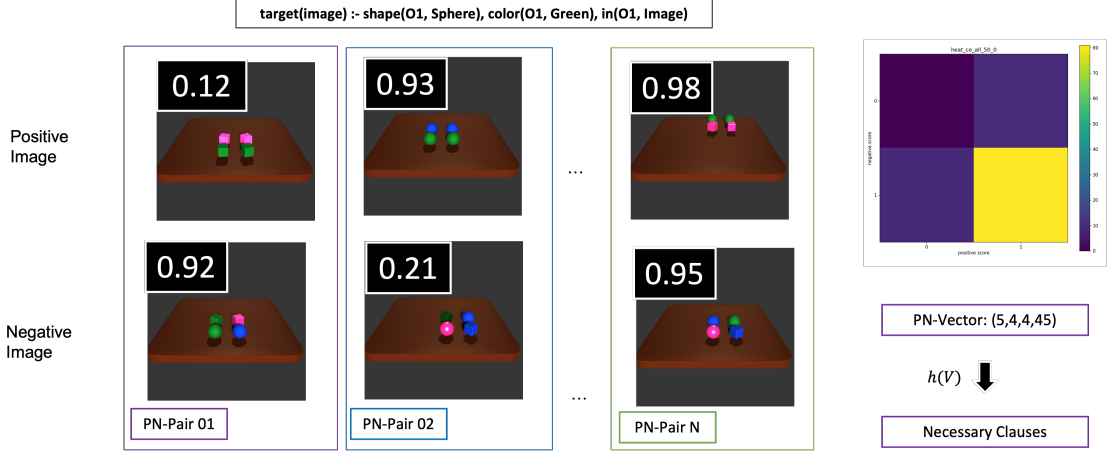


Figure 4: Left: PN pairs. Middle: PN pair scores on coordinate system. The points are clustered in four corners. These scores are evaluated by NSFR. Right: Four Score Map. The four score map illustrates the result of 4 kinds of evaluation on one clause, i.e. high positive-high negative, high positive-low negative, low positive-high negative, low positive-low negative.

We prepare equal number of positive and negative images for evaluation, thus all the images can be paired from two groups. A new generated clause is evaluated on all the PN pairs. Evaluation on each pair getting two values, one from positive and one from negative. We can take negative value as x axis, positive value as y axis and draw all the evaluation result on a coordinate system. Thus each PN pair corresponds a point. We observed these points are only appears in the four clusters in the coordinate system(as show in figure 4 middle.), thus we fuzzy these points to only four areas, each take one cluster. Thus for these two values, we only have $2^2 = 4$ different combinations. We named it as *four score map*.

A **four score map** fuzzy the evaluation result on an image to positive 1 and negative 0, map the positive image result and negative image result to four areas, namely (0, 0), (0, 1), (1, 0) and (1, 1). We found that four score map has good description for sufficient and necessary conditions in logic.

Base on the scoring areas of the predicates, we can classify them into several groups. Let N denotes the number of PN pairs.

Definition 3.2 (Sufficient and Necessary Clause). Scores on (0, 1) only, i.e. $s_{01} = N$ They are sufficient and necessary for the target pattern.

Definition 3.3 (Necessary Clause). Scores on (0, 1) and (1, 1), i.e. $s_{01} + s_{11} = N$. They are necessary for the target pattern. Note that they always true in the positive images, but also can be true in negative images.

Definition 3.4 (Sufficient Clause). Scores on (0, 0) and (0, 1), $s_{00} + s_{01} = N$ and $s_{01} > 0$. It induces directly some of positive patterns but can be failed on some other positive patterns. It never induces any negative patterns.

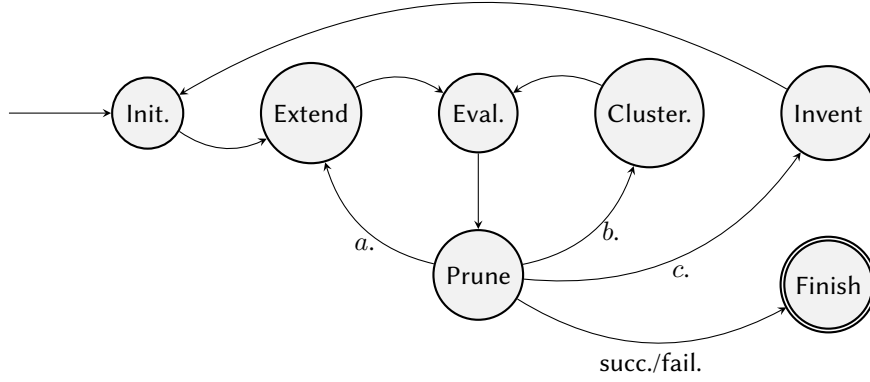


Figure 5: The workflow of NeSy- π . Edge a fires when maximum extension time is not exceed or no new predicates are invented after clustering. Edge b fires when maximum extension times is exceed. Edge c fires when new predicates are kept after pruning.

The key idea of these definitions is to provide an evaluation metric for searched clauses. Since it is impossible to consider all the clauses for rule searching, an efficient pruning strategy is necessary. Our pruning strategy is mainly based on the satisfaction of the clauses on these definitions.

The workflow of NeSy- π is shown in figure 5. The system’s work is basically to maintain a clause set \mathcal{C} . We add promising clauses inside and remove the abandoned ones from it. We explain each state as follows

Initial The searching is started from here. The input is the most general clause, i.e. the one that only states the existence of objects in the scene. For a scene with at most 2 objects, the initial clause set is $\mathcal{C} = \{\text{target}(X) : -\text{in}(01, X), \text{in}(02, X).\}$. Note that initial clause is a necessary clause since it is true on all the positive images.

Extend The input is either the initial clause set or the clauses set from pruning. For each clause in the set, we extend it with one predicate. Since there are many possible predicates to chose, any of them can be a promising one, thus we consider all the extension possibility into account. For example, to extend clause c , if there are 10 predicates in the language, the number of extended clauses of c is up to 10, if all the extended clauses are consistent with itself.

Evaluate In order to find all the promising clauses, we evaluate all the extended new clauses or clustered clauses. For each evaluation unit, we evaluate it on the whole dataset and classify it based on its four score map.

Pruning Pruning is based on scores of each clause or clause cluster getting from evaluate state. A prune strategy is required in this step. In this paper, we select only top ranked necessary/sufficient clauses/clusters, the rests are pruned.

Cluster The cluster is one step as preparation of high level concepts invention. For example, if blue, red are the only appearing colors in all the scenes, the concept `two_objs_with_same_color` can be clustered by clauses

```
two_objs_with_same_color(X):-in(01,X),in(02,X),color(01,Blue),color(02,blue).
two_objs_with_same_color(X):-in(01,X),in(02,X),color(01,Red),color(02,red).
```

After pruning, if the max iteration of clause extension is exceed, the system will cluster the clauses. The input is clause set \mathcal{C} . We calculate all the subsets of \mathcal{C} . Each subset is considered as one cluster. The number of clusters is up to 2^n , where $n = |\mathcal{C}|$ i.e. the size of clause set. Since it grows exponentially, for a large n , we cannot evaluate all the subset. In order to improve the computation efficiency, the size of subset can be constrained with a threshold, to control the maximum size of the subset.

Inventing The input of inventing state is the set of clause clusters. Each cluster is considered as a new predicate, where its body is the clause clusters. After invention, the new predicates update the language and the system goes to initial state again.

Finish The system finish the rule searching if any sufficient and necessary predicate or clause is found or failed by exceeding the maximum iterations.

4. Experimental Evaluation

We aim to answer the following questions:

- Q1:** Does NeSy- π invent predicates from complex visual scenes for NeSy ILP systems to reduce required background knowledge?
- Q2:** Is NeSy- π computationally efficient?
- Q3:** Can 3D Kandinsky patterns provide efficient evaluations for NeSy-ILP systems?

To solve 2D Kandinsky Patterns, we provide following constants as background knowledge, shape: square/triangle/circle; color: red/yellow/blue; distance: from 0 to the maximum distance (the image width), we divide it to 8 levels, denote by $\rho_1/\dots/\rho_8$, where ρ_1 denote the distance within 0 pixel, ρ_8 corresponds the distance equal to the width of the image, others denote something in between; direction: similar to distance, the direction is represented by 8 predicates $\phi_1/\dots/\phi_8$ where each one corresponds $360/8 = 45^\circ$ of direction. [\[We need a visual explanation for this -HK\]](#) No further information is given, such as different color, different shape, etc. Fig. 6 shows some of the patterns in 2DKP.

To solve 3D Kandinsky Patterns, we provide following constants as background knowledge, shape: sphere/cube; color: pink/green/blue; distance: all the positions of 3D objects are normalized. 8 distance predicates are used to represent 8 levels of distance evenly; direction: we assume all the object place on a flat surface with similar height, thus the direction between two objects can be represented by two values only. We divide the direction into 8 levels evenly where each one corresponds 45° of directions. 6 shows some of the patterns in 3DKP. The learning result on each patterns in Kandinsky pattern is shown in table 2. Fig. 4 shows the learned clause and invented predicates of *cross* pattern in 3DKP.

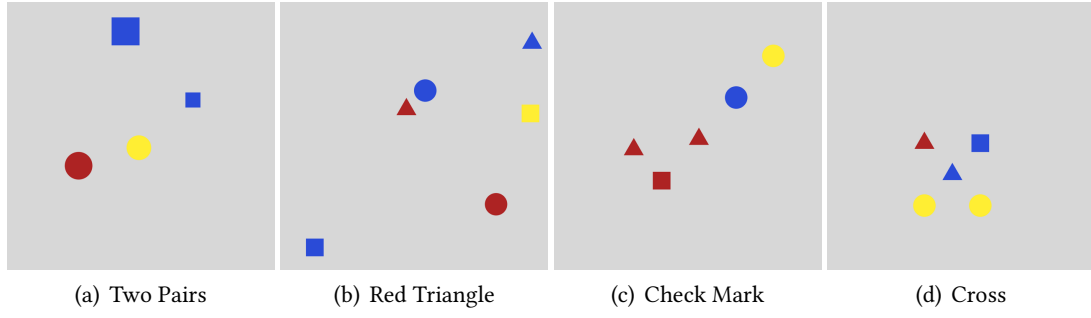


Figure 6: Kandinsky Patterns. **Two Pairs:** This pattern always have 4 objects inside, two of them has same color and same shape, the other two has same shape and different color. **Red Triangle:** This pattern always have a red triangle, and another object with different color and different shape nearby, the rest 4 objects are random objects. **Check Mark:** the positions of 5 objects consists of an outline of check mark. **Cross:** the position of 5 objects consists of an outline of a cross mark.

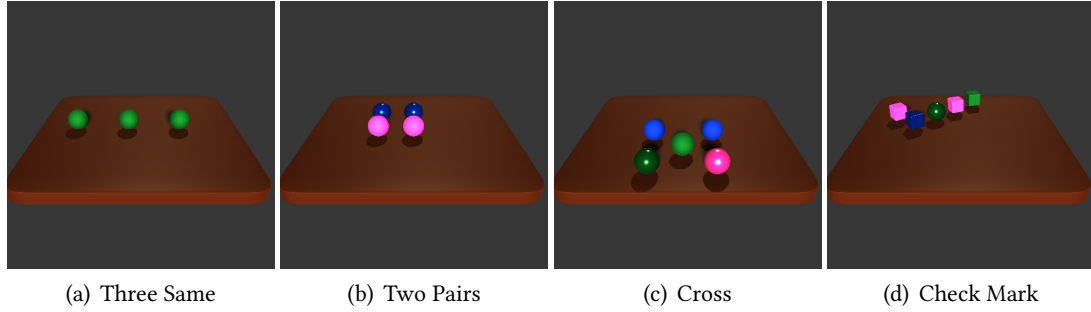


Figure 7: 3D Kandinsky Patterns. **Three Same:** This pattern always have 3 objects inside, three of them has same color and same shape. **Two Pairs:** This pattern always have 4 objects inside, two of them has same color and same shape. The other two also have same color and same shape. **Cross Mark:** this pattern always have 5 objects with same shape inside, the positions of 5 objects consists of an outline of cross mark. **Check Mark:** this pattern always have 5 objects, the position of 5 objects consists of an outline of a check mark.

5. Related Work

Inductive Logic Programming [?] has emerged at the intersection of machine learning and logic programming. Many ILP frameworks have been developed, e.g., FOIL [?], Progol [?], ILASP [?], Metagol [?], and Popper [?].

Although predicate invention has been proposed since 1988 by [1], it is always be considered as a major challenge. Most ILP system do not supports PI, including classic systems like Progol [2], TILDE[3] and modern system such as ATOM [4], LFIT [5].

Some works have been proposed for PI systems. Predicate Invention (PI) has been addressed [?] [?] [?] [?] [?] [?]. [TODO: split -HK] However, none of these are focus on visual images and learning logical patterns existing in the visual scenes. [6] is an approach for predicate invention, which generate constrains from inconsistent hypothesis and further be used for pruning all

```

target(X):-in(01,X),in(02,X),in(03,X),in(04,X),in(05,X),
inv_pred15(01,02,03,04,05).

inv_pred15(01,02,03,04,05):-in(01,X),in(02,X),in(03,X),in(04,X),in(05,X),
inv_pred6(02,03,04,05),shape(01,cube),shape(02,cube),shape(03,cube).

inv_pred15(01,02,03,04,05):-in(01,X),in(02,X),in(03,X),in(04,X),in(05,X),
inv_pred6(02,03,04,05),shape(01,sphere),shape(02,sphere),shape(03,sphere).

inv_pred6(01,03,04,05):-in(01,X),in(03,X),in(04,X),in(05,X),
inv_pred3(01,03,04,05),phi(03,05,phi6),rho(03,05,rho1).

inv_pred3(02,03,04,05):-in(02,X),in(03,X),in(04,X),in(05,X), rho(02,05,rho2),
rho(03,04,rho2).

```

Figure 8: Learned rules by NeSy- π on Cross pattern in 3D Kandinsky Pattern scenes.

Table 2

Experiment result on 2D Kandinsky Patterns and 3D Kandinsky Patterns. The dataset of each experiment has 64 PN pairs. Batch size is 8. All the experiments are tested on a single NVIDIA A100 GPU.

Dataset	Patterns	Iterations	Times		Accuracy		# of Ne-Preds	
			α ILP	NeSy- π	α ILP	NeSy- π	α ILP	NeSy- π
2D KP	red-triangle	3	0				0	4
	two-pairs	2	0				0	3
	check-mark		0				0	
	cross-mark		0				0	
3D KP	three same	2	0				0	1
	two-pairs	2	0				0	3
	cross mark	2	0				0	3
	check mark	1	0				0	2

the similar clauses. Several system [8], [9] uses meta-rules as templates for new predicates invention. α ILP (citation?) supports visual image as input but require background knowledge during the reasoning.

6. Conclusion

In this paper, we proposed an approach for Neural-Symbolic Predicate Invention. NeSy-PI is able to find the new knowledge and summarize it as new predicates, thus it requires less background knowledge for reasoning. In our experiment, we show that our PI model can successfully find the target program given only basic neural perception results and relevant constants, no further background knowledge is required. We also show our efficient prune strategy for predicate searching, the searching result is acquired faster and still sound.

A. Experiment details on 3DKP

(<pre> three same) kp(X):-in(01,X),in(02,X),in(03,X),inv_pred2(01,02,03). inv_pred2(01,02,03):-color(01,blue),color(02,blue),color(03,blue), in(01,X),in(02,X),in(03,X),shape(01,cube),shape(02,cube),shape(03,cube). inv_pred2(01,02,03):-color(01,blue),color(02,blue),color(03,blue),in(01,X), in(02,X),in(03,X),shape(01,sphere),shape(02,sphere),shape(03,sphere). inv_pred2(01,02,03):-color(01,green),color(02,green),color(03,green),in(01,X), in(02,X),in(03,X),shape(01,cube),shape(02,cube),shape(03,cube). inv_pred2(01,02,03):-color(01,green),color(02,green),color(03,green),in(01,X), in(02,X),in(03,X),shape(01,sphere),shape(02,sphere),shape(03,sphere). inv_pred2(01,02,03):-color(01,pink),color(02,pink),color(03,pink),in(01,X), in(02,X),in(03,X),shape(01,cube),shape(02,cube),shape(03,cube). inv_pred2(01,02,03):-color(01,pink),color(02,pink),color(03,pink),in(01,X), in(02,X),in(03,X),shape(01,sphere),shape(02,sphere),shape(03,sphere). </pre>
(<pre> two pairs) kp(X):-in(01,X),in(02,X),in(03,X),in(04,X),inv_pred30(01,02,03,04), inv_pred7(03,04). inv_pred30(01,02,03,04):-in(01,X),in(02,X),in(03,X),in(04,X),inv_pred2(01,02,03,04), inv_pred7(01,02). inv_pred7(01,02):-in(01,X),in(02,X),shape(01,cube),shape(02,cube). inv_pred7(01,02):-in(01,X),in(02,X),shape(01,sphere),shape(02,sphere). inv_pred2(01,02):-color(01,blue),color(02,blue),in(01,X),in(02,X). inv_pred2(01,02):-color(01,green),color(02,green),in(01,X),in(02,X). inv_pred2(01,02):-color(01,pink),color(02,pink),in(01,X),in(02,X). </pre>
(<pre> cross mark) kp(X):-in(01,X),in(02,X),in(03,X),in(04,X),in(05,X),inv_pred15(01,02,03,04,05), inv_pred3(02,03,04,05):-in(02,X),in(03,X),in(04,X),in(05,X),rho(02,05,rho2), rho(03,04,rho2). inv_pred6(01,03,04,05):-in(01,X),in(03,X),in(04,X),in(05,X), inv_pred3(01,03,04,05),phi(03,05,phi6),rho(03,05,rho1). inv_pred15(01,02,03,04,05):-in(01,X),in(02,X),in(03,X),in(04,X),in(05,X), inv_pred6(02,03,04,05),shape(01,cube),shape(02,cube),shape(03,cube). inv_pred15(01,02,03,04,05):-in(01,X),in(02,X),in(03,X),in(04,X),in(05,X), inv_pred6(02,03,04,05),shape(01,sphere),shape(02,sphere),shape(03,sphere). </pre>
(<pre> check mark) kp(X):-in(01,X),in(02,X),in(03,X),in(04,X),in(05,X),inv_pred3(01,02,03,05), inv_pred4(02,04,05). inv_pred3(01,02,03,05):-in(01,X),in(02,X),in(03,X),in(05,X),rho(01,02,rho0), rho(03,05,rho0). inv_pred4(01,02,05):-in(01,X),in(02,X),in(05,X),rho(01,02,rho0),rho(01,05,rho0). </pre>

Figure 9: Learned rules by NeSy- π on 3D Kandinsky Pattern scenes.

References

- [1] Machine invention of first-order predicates by inverting resolution, in: J. Laird (Ed.), Machine Learning Proceedings 1988, Morgan Kaufmann, San Francisco (CA), 1988, pp. 339–352. URL: <https://www.sciencedirect.com/science/article/pii/B9780934613644500402>. doi:<https://doi.org/10.1016/B978-0-934613-64-4.50040-2>.

- [2] S. H. Muggleton, Inverse entailment and prolog, *New Generation Computing* 13 (1995) 245–286.
- [3] H. Blockeel, L. De Raedt, Top-down induction of first-order logical decision trees, *Artificial Intelligence* 101 (1998) 285–297. URL: <https://www.sciencedirect.com/science/article/pii/S0004370298000344>. doi:[https://doi.org/10.1016/S0004-3702\(98\)00034-4](https://doi.org/10.1016/S0004-3702(98)00034-4).
- [4] J. Ahlgren, S. Y. Yuen, Efficient program synthesis using constraint satisfaction in inductive logic programming, *J. Mach. Learn. Res.* 14 (2013) 3649–3682.
- [5] K. Inoue, T. Ribeiro, C. Sakama, Learning from interpretation transition, *Machine Learning* 94 (2014). doi:10.1007/s10994-013-5353-8.
- [6] A. Cropper, R. Morel, Predicate invention by learning from failures, 2021. URL: <https://arxiv.org/abs/2104.14426>. doi:10.48550/ARXIV.2104.14426.
- [7] D. M. Cerna, A. Cropper, Generalisation through negation and predicate invention, 2023. URL: <https://arxiv.org/abs/2301.07629>. doi:10.48550/ARXIV.2301.07629.
- [8] R. Evans, E. Grefenstette, Learning explanatory rules from noisy data, *CoRR* abs/1711.04574 (2017). URL: <http://arxiv.org/abs/1711.04574>. arXiv:1711.04574.
- [9] T. KAMINSKI, T. EITER, K. INOUE, Exploiting answer set programming with external sources for meta-interpretive learning, *Theory and Practice of Logic Programming* 18 (2018) 571–588. doi:10.1017/S1471068418000261.
- [10] R. Evans, E. Grefenstette, Learning explanatory rules from noisy data (extended abstract), in: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, International Joint Conferences on Artificial Intelligence Organization*, 2018, pp. 5598–5602. URL: <https://doi.org/10.24963/ijcai.2018/792>. doi:10.24963/ijcai.2018/792.
- [11] L. Lamport, *TEX: A Document Preparation System*, Addison-Wesley, Reading, MA., 1986.
- [12] P. S. Abril, R. Plant, The patent holder’s dilemma: Buy, sell, or troll?, *Communications of the ACM* 50 (2007) 36–44. doi:10.1145/1188913.1188915.
- [13] S. Cohen, W. Nutt, Y. Sagic, Deciding equivalences among conjunctive aggregate queries, *J. ACM* 54 (2007). doi:10.1145/1219092.1219093.
- [14] J. Cohen (Ed.), Special issue: Digital Libraries, volume 39, 1996.
- [15] D. Kosiur, *Understanding Policy-Based Networking*, 2nd. ed., Wiley, New York, NY, 2001.
- [16] D. Harel, *First-Order Dynamic Logic*, volume 68 of *Lecture Notes in Computer Science*, Springer-Verlag, New York, NY, 1979. doi:10.1007/3-540-09237-4.
- [17] I. Editor (Ed.), The title of book one, volume 9 of *The name of the series one*, 1st. ed., University of Chicago Press, Chicago, 2007. doi:10.1007/3-540-09237-4.
- [18] I. Editor (Ed.), The title of book two, The name of the series two, 2nd. ed., University of Chicago Press, Chicago, 2008. doi:10.1007/3-540-09237-4.
- [19] A. Z. Spector, Achieving application requirements, in: S. Mullender (Ed.), *Distributed Systems*, 2nd. ed., ACM Press, New York, NY, 1990, pp. 19–33. doi:10.1145/90417.90738.
- [20] B. P. Douglass, D. Harel, M. B. Trakhtenbrot, Statecharts in use: structured analysis and object-orientation, in: G. Rozenberg, F. W. Vaandrager (Eds.), *Lectures on Embedded Systems*, volume 1494 of *Lecture Notes in Computer Science*, Springer-Verlag, London, 1998, pp. 368–394. doi:10.1007/3-540-65193-4_29.
- [21] D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms* (3rd. ed.), Addison Wesley Longman Publishing Co., Inc., 1997.

- [22] S. Andler, Predicate path expressions, in: Proceedings of the 6th. ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages, POPL '79, ACM Press, New York, NY, 1979, pp. 226–236. doi:10.1145/567752.567774.
- [23] S. W. Smith, An experiment in bibliographic mark-up: Parsing metadata for xml export, in: R. N. Smythe, A. Noble (Eds.), Proceedings of the 3rd. annual workshop on Librarians and Computers, volume 3 of LAC '10, Paparazzi Press, Milan Italy, 2010, pp. 422–431. doi:99.9999/woot07-S422.
- [24] M. V. Gundy, D. Balzarotti, G. Vigna, Catch me, if you can: Evading network signatures with web-based polymorphic worms, in: Proceedings of the first USENIX workshop on Offensive Technologies, WOOT '07, USENIX Association, Berkley, CA, 2007.
- [25] D. Harel, LOGICS of Programs: AXIOMATICS and DESCRIPTIVE POWER, MIT Research Lab Technical Report TR-200, Massachusetts Institute of Technology, Cambridge, MA, 1978.
- [26] K. L. Clarkson, Algorithms for Closest-Point Problems (Computational Geometry), Ph.D. thesis, Stanford University, Palo Alto, CA, 1985. UMI Order Number: AAT 8506171.
- [27] D. A. Anisi, Optimal Motion Control of a Ground Vehicle, Master's thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2003.
- [28] H. Thornburg, Introduction to bayesian statistics, 2001. URL: <http://ccrma.stanford.edu/~jos/bayes/bayes.html>.
- [29] R. Ablamowicz, B. Fauser, Clifford: a maple 11 package for clifford algebra computations, version 11, 2007. URL: <http://math.tntech.edu/rafal/cliff11/index.html>.
- [30] Poker-Edge.Com, Stats and analysis, 2006. URL: <http://www.poker-edge.com/stats.php>.
- [31] B. Obama, A more perfect union, Video, 2008. URL: <http://video.google.com/videoplay?docid=6528042696351994555>.
- [32] D. Novak, Solder man, in: ACM SIGGRAPH 2003 Video Review on Animation theater Program: Part I - Vol. 145 (July 27–27, 2003), ACM Press, New York, NY, 2003, p. 4. URL: <http://video.google.com/videoplay?docid=6528042696351994555>. doi:99.9999/woot07-S422.
- [33] N. Lee, Interview with bill kinder: January 13, 2005, Comput. Entertain. 3 (2005). doi:10.1145/1057270.1057278.
- [34] J. Scientist, The fountain of youth, 2009. Patent No. 12345, Filed July 1st., 2008, Issued Aug. 9th., 2009.
- [35] B. Rous, The enabling of digital libraries, Digital Libraries 12 (2008). To appear.
- [36] M. Saeedi, M. S. Zamani, M. Sedighi, A library-based synthesis methodology for reversible logic, Microelectron. J. 41 (2010) 185–194.
- [37] M. Saeedi, M. S. Zamani, M. Sedighi, Z. Sasanian, Synthesis of reversible circuit using cycle-based approach, J. Emerg. Technol. Comput. Syst. 6 (2010).
- [38] M. Kirschmer, J. Voight, Algorithmic enumeration of ideal classes for quaternion orders, SIAM J. Comput. 39 (2010) 1714–1747. URL: <http://dx.doi.org/10.1137/080734467>. doi:10.1137/080734467.
- [39] L. Hörmander, The analysis of linear partial differential operators. IV, volume 275 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*, Springer-Verlag, Berlin, Germany, 1985. Fourier integral operators.
- [40] L. Hörmander, The analysis of linear partial differential operators. III, volume 275 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*, Springer-Verlag, Berlin, Germany, 1985. Fourier integral operators.

- Sciences*], Springer-Verlag, Berlin, Germany, 1985. Pseudodifferential operators.
- [41] IEEE, Ieee tcsc executive committee, in: Proceedings of the IEEE International Conference on Web Services, ICWS '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 21–22. doi:10.1109/ICWS.2004.64.
 - [42] TUG, Institutional members of the T_EX users group, 2017. URL: <http://www.tug.org/instmemb.html>.
 - [43] R Core Team, R: A language and environment for statistical computing, 2019. URL: <https://www.R-project.org/>.
 - [44] S. Anzaroot, A. McCallum, UMass citation field extraction dataset, 2013. URL: <http://www.iesl.cs.umass.edu/data/data-umasscitationfield>.