Abductive Knowledge Induction From Raw Data

Wang-Zhou Dai , Stephen H. Muggleton
Department of Computing, Imperial College London, London, UK
{w.dai, s.muggleton}@imperial.ac.uk

Abstract

For many reasoning-heavy tasks involving raw inputs, it is challenging to design an appropriate end-to-end learning pipeline. Neuro-Symbolic Learning, divide the process into sub-symbolic perception and symbolic reasoning, trying to utilise data-driven machine learning and knowledge-driven reasoning simultaneously. However, they suffer from the exponential computational complexity within the interface between these two components, where the sub-symbolic learning model lacks direct supervision, and the symbolic model lacks accurate input facts. Hence, most of them assume the existence of a strong symbolic knowledge base and only learn the perception model while avoiding a crucial problem: where does the knowledge come from? In this paper, we present Abductive Meta-Interpretive Learning ($Meta_{Abd}$) that unites abduction and induction to learn neural networks and induce logic theories jointly from raw data. Experimental results demonstrate that $Meta_{Abd}$ not only outperforms the compared systems in predictive accuracy and data efficiency but also induces logic programs that can be re-used as background knowledge in subsequent learning tasks. To the best of our knowledge, $Meta_{Abd}$ is the first system that can jointly learn neural networks from scratch and induce recursive first-order logic theories with predicate invention.

1 Introduction

Despite the success of data-driven end-to-end deep learning in many traditional machine learning tasks, it has been shown that incorporating domain knowledge is still necessary for some complex learning problems [Dhingra *et al.*, 2020; Grover *et al.*, 2019; Trask *et al.*, 2018]. In order to leverage complex domain knowledge that is discrete and relational, end-to-end learning systems need to represent it with a differentiable module that can be embedded in the deep learning context. For example, graph neural networks (GNN) use relational graphs as an external knowledge base [Zhou *et al.*, 2018]; some works even considers more specific domain knowledge such as differentiable primitive predicates and programs [Dong *et al.*, 2019; Gaunt *et al.*, 2017]. However, it is hard to design a unified differentiable module to accurately represent general relational knowledge, which may contain complex inference structures such as recursion [Glasmachers, 2017; Garcez *et al.*, 2019].

Therefore, many researchers propose to break the end-to-end learning pipeline apart, and build a hybrid model that consists of smaller modules where each of them only accounts for one specific function [Glasmachers, 2017]. A representative branch in this line of research is Neuro-Symbolic (NeSy) AI [De Raedt *et al.*, 2020; Garcez *et al.*, 2019] aiming to bridge System 1 and System 2 AI [Kahneman, 2011; Bengio, 2017], i.e., neural-network-based machine learning and symbolic-based relational inference.

However, the lack of supervision in the non-differentiable interface between neural and symbolic systems, based on the facts extracted from raw data and their truth values, leads to high computational complexity in learning [Li *et al.*, 2020; Dai *et al.*, 2019]. Consequently, almost all neural-symbolic models assume the existence of a very strong predefined domain knowledge base and could not perform program induction. This limits the expressive power of the hybrid-structured model and sacrifices many benefits of symbolic learning (e.g., predicate invention, learning recursive theories, and re-using learned models as background knowledge).

In this paper, we integrate neural networks with Inductive Logic Programming (ILP) [Muggleton and de Raedt, 1994] to enable first-order logic theory induction from raw data. More specifically, we present Abductive Meta-Interpretive Learning ($Meta_{Abd}$) which extends the Abductive Learning (ABL) framework [Dai *et al.*, 2019; Zhou, 2019] by combining logical induction and abduction [Flach *et al.*, 2000] with neural networks in Meta-Interpretive Learning (MIL) [Muggleton *et al.*, 2015]. $Meta_{Abd}$ employs neural networks to extract probabilistic logic facts from raw data, and induces an abductive logic program [Kakas *et al.*, 1992] that can efficiently infer the truth values of the facts to train the neural model.

To the best of our knowledge, $Meta_{Abd}$ is the first system that can simultaneously (1) train neural models from scratch, (2) learn recursive logic theories and (3) perform predicate invention from domains with sub-symbolic representation. In the experiments we compare $Meta_{Abd}$ to the compared state-of-the-art end-to-end deep learning models and neuro-symbolic methods on two complex learning tasks. The results

show that, given the same amount of background knowledge, $Meta_{Abd}$ outperforms the compared models significantly in terms of predictive accuracy and data efficiency, and learns human interpretable models that could be re-used in subsequent learning tasks.

2 Related Work

Solving "System 2" problems requires the ability of relational and logical reasoning [Kahneman, 2011; Bengio, 2017]. Due to its complexity, many researchers have tried to embed intricate background knowledge in end-to-end deep learning models. For example, [Trask *et al.*, 2018] propose the differentiable Neural Arithmetic Logic Units (NALU) to model basic arithmetic functions (e.g., addition, multiplication, etc.) in neural cells; [Grover *et al.*, 2019] encode permutation operators with a stochastic matrix and present a differentiable approximation to the sort operation; [Wang *et al.*, 2019] introduce a differentiable SAT solver to enable gradient-based constraint solving. However, most of these specially designed differentiable modules are *ad hoc* approximations to the original symbolic inference mechanisms.

To exploit the complex background knowledge expressed by formal languages directly, Statistical Relational (StarAI) and Neural Symbolic (NeSy) AI [De Raedt *et al.*, 2020; Garcez *et al.*, 2019] try to use probabilistic inference or other differentiable functions to approximate logical inference [Cohen *et al.*, 2020; Dong *et al.*, 2019; Manhaeve *et al.*, 2018; Donadello *et al.*, 2017]. However, they require a pre-defined symbolic knowledge base and only train the attached neural/probabilistic models due to the highly complex interface between the neural and symbolic modules.

One way to learn symbolic theories is to use Inductive Logic Programming [Muggleton and de Raedt, 1994]. Some early work on combining logical abduction and induction can learn logic theories even when input data is incomplete [Flach *et al.*, 2000]. Recently, ∂ILP was proposed for learning first-order logic theories from noisy data [Evans and Grefenstette, 2018]. However, ILP-based works are designed for learning in symbolic domains. Otherwise, they need to use a fully trained neural models to make sense of the raw inputs.

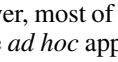
Machine apperception [Evans *et al.*, 2021] unifies Answer Set Programming with perception by modelling it with binary neural networks. It can learn recursive logic theories and perform concept (monadic predicate) invention. However, both logic hypotheses and the parameters of neural networks are represented by logical groundings, making the system very hard to optimise. For problems involving noisy inputs like MNIST images, it still requires a fully pre-trained perceptual neural net to extract the symbols like ILP-based methods.

Previous work on Abductive Learning (ABL) [Dai *et al.*, 2019; Dai and Zhou, 2017] also unites sub-symbolic perception and symbolic reasoning through abduction, but they need a pre-defined knowledge base to enable abduction and cannot perform program induction. Our presented Abductive Meta-Interpretive Learning takes a step further, which not only learns a perception model that can make sense of raw data, but also learns logic programs and performs predicate invention to understand the underlying relations in the task.

3 Abductive Meta-Interpretive Learning

3.1 Problem Formulation

A typical model bridging sub-symbolic and symbolic learning contains two major parts: a perception model and a reasoning model [Dai *et al.*, 2019]. The perception model maps sub-symbolic inputs $x \in \mathcal{X}$ to some primitive symbols $z \in \mathcal{Z}$, such as digits, objects, ground logical expressions, etc. The reasoning model takes the interpreted z as input and infers the final output $y \in \mathcal{Y}$ according to a symbolic knowledge base B. Because the primitive symbols z are *uncertain* and *not observable* from both training data and the knowledge base, we have named them as *pseudo-labels* of x.

The perception model is parameterised with θ and outputs the conditional probability $P_\theta(z|x) = P(z|x, \theta)$; the reasoning model $H \in \mathcal{H}$ is a set of *first-order* logical clauses such that $B \cup H \cup z \models y$, where "$\models$" means "logically entails". Our target is to learn θ and H simultaneously from training data $D = \{\langle x_i, y_i\rangle\}_{i=1}^n$. For example, if we have one example with $x =$  and $y = 6$, given background knowledge about adding two numbers, the hybrid model should learn a perception model that recognises $z = [1, 2, 3]$ and induce a program to add each number in z recursively.

Assuming that D is an i.i.d. sample from the underlying distribution of (x, y), our objective can be represented as

$$(H^*, \theta^*) = \arg\max_{H,\theta} \prod_{\langle x,y\rangle \in D} \sum_{z \in \mathcal{Z}} P(y, z|B, x, H, \theta), \quad (1)$$

where pseudo-label z is a hidden variable. Theoretically, this problem can be solved by Expectation Maximisation (EM) algorithm. However, the symbolic hypothesis H—a first-order logic theory—is difficult to be optimised together with the parameter θ, which has a continuous hypothesis space.

We propose to solve this problem by treating H like z as an extra hidden variable, which gives us:

$$\theta^* = \arg\max_{\theta} \prod_{\langle x,y\rangle \in D} \sum_{H \in \mathcal{H}} \sum_{z \in \mathcal{Z}} P(y, H, z|B, x, \theta). \quad (2)$$

Now, the learning problem can be split into two EM steps: (1) **Expectation:** obtaining the expected value of H and z by sampling them in their discrete hypothesis space from $(H, z) \sim P(H, z|B, x, y, \theta)$; (2) **Maximisation:** estimating θ by maximising the likelihood of training data with numerical optimisation approaches such as gradient descent.

Challenges. The main challenge is to estimate the expectation of the hidden variables $H \cup z$, i.e., we need to search for the most probable H and z given the θ learned in the previous iteration. This is not trivial. Even when B is sound and complete, estimating the truth-values of hidden variable z results in a search space growing exponentially with the number of training examples, which is verified in our experiments with DeepProblog [Manhaeve *et al.*, 2018] in section 4.1.

Furthermore, H and z are entangled, which makes the tasks even harder. For example, given $x =$ and $y = 6$, when the perception model correctly recognises $z = [1, 2, 3]$, we have at least two choices for H: cumulative sum or cumulative product. When the perception model is under-trained and outputs $z = [2, 2, 3]$, then H could be a program that only multiplies the last two digits.

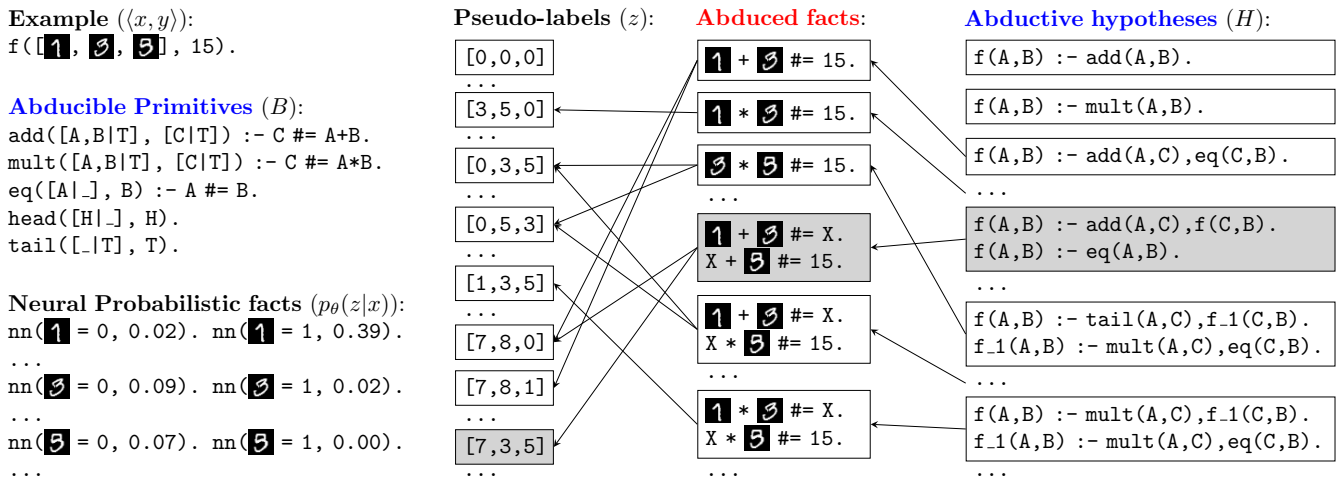


Figure 1: Example of $Meta_{Abd}$'s abduction-induction learning. Given training examples, background knowledge of abducible primitives and probabilistic facts generated by a perceptual neural net, $Meta_{Abd}$ learns an abductive logic program H and abduces relational facts as constraints (implemented with the CLP(Z) predicate "#="[1]) over the input images; it then uses them to efficiently prune the search space of the most probable pseudo-labels z (in grey blocks) for training the neural network.

3.2 Probabilistic Abduction-Induction Reasoning

Inspired by early works in abductive logic programming [Flach *et al.*, 2000], we propose to solve the challenges above by combining logical induction and abduction. The induction learns an abductive logic theory H based on $P_\theta(z|x)$; the abduction made by H reduces the search space of z.

Abductive reasoning, or *abduction* refers to the process of selectively inferring specific grounded facts and hypotheses that give the best explanation to observations based on background knowledge of a deductive theory.

Definition 3.1 (Abducible primitive) *An abducible primitive is a predicate that defines the explanatory grounding facts in abductive reasoning.*

Definition 3.2 (Abductive hypothesis) *An abductive hypothesis is a set of first-order logic clauses whose body contains literals of abductive primitives.*

Following is an example of using *abductive hypothesis* and *abducible primitive* in problem-solving:

Example 1 Observing raw inputs $x = [2, 3, 1]$ and a symbolic output $y = 6$, we could formulate an *abductive hypothesis* H that is a recursive cumulative sum function, whose *abductive primitives* are "+" and "=". Hence, H will abduce a set of explanatory ground facts $\{2 + 3 = \mathtt{Z}, \mathtt{Z} + 1 = 6\}$. Based on these facts, we could infer that none of the digits in x is greater than 6. Furthermore, if the current perception model assigns very high probabilities to $2 = 2$ and $3 = 3$, we could easily infer that $1 = 1$ even when the perception model has relatively low confidence about it.

An illustrative example of combining abduction and induction with probabilities is shown in Fig. 1. Briefly speaking, instead of directly sampling pseudo-labels z and H together from the huge hypothesis space, our $Meta_{Abd}$ induces *abductive hypothesis* H consists of *abducible primitives*, and then use the abduced facts to prune the search space of z. Meanwhile, the perception model outputs the likelihood of pseudo-labels with $p_\theta(z|x)$ defining a distribution over all possible values of z and helps to find the most probable $H \cup z$.

Formally, we re-write the likelihood of each $\langle x, y \rangle$ in Eq. 2:

$$\begin{aligned} P(y, H, z|B, x, \theta) &= P(y, H|B, z)P_\theta(z|x) \\ &= P(y|B, H, z)P(H|B, z)P_\theta(z|x) \\ &= P(y|B, H, z)P_{\sigma^*}(H|B)P_\theta(z|x), \end{aligned} \quad (3)$$

where $P_{\sigma^*}(H|B)$ is the Bayesian prior distribution on first-order logic hypotheses, which is defined by the transitive closure of *stochastic refinements* σ^* given the background knowledge B [Muggleton *et al.*, 2013], where a refinement σ is a unit modification (e.g., adding/removing a clause or literal) to a logic theory. The equations hold because: (1) pseudo-label z is conditioned on x and θ since it is the output of the perception model; (2) H follows the prior distribution so it only depends on B; (3) $y \cup H$ is independent from x given z because the relations among B, H, y and z are determined by pure logical inference, where:

$$P(y|B, H, z) = \begin{cases} 1, & \text{if } B \cup H \cup z \models y, \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

Following Bayes' rule we have $P(H, z|B, x, y, \theta) \propto P(y, H, z|B, x, \theta)$. Now we can search for the most probable $H \cup z$ in the expectation step according to Eq. 3 as follows:

1. Induce an *abductive* theory $H \sim P_{\sigma^*}(H|B)$;
2. Use $H \cup B$ and y to *abduce*[2] possible pseudo-labels z, which are *guaranteed* to satisfy $H \cup B \cup z \vdash y$ and exclude the values of z such that $P(y|B, H, z) = 0$;

[1] CLP(Z) is a constraint logic programming package accessible at https://github.com/triska/clpz. More implementation details are in the appendix.

[2] It can be parallelled, please see the appendix.

Abductive Meta-Interpreter

```
prove([], Prog, Prog, [], Prob, Prob).
prove([Atom|As], Prog1, Prog1, Abds, Prob1, Prob2) :-
     deduce(Atom),
     prove(As, Prog1, Prog2, Abds, Prob1, Prob2).
prove([Atom|As], Prog1, Prog1, Abds, Prob1, Prob2) :-
     call_abducible(Atom, Abd, Prob),
     Prob3 is Prob1 * Prob,
     get_max_prob(Max), Prob3 > Max,
     set_max_prob(Prob3),
     prove(As, Prog1, Prog1, [Abd|Abds], Prob3, Prob2).
prove([Atom|As], Prog1, Prog2, Abds, Prob1, Prob2) :-
     meta-rule(Name, MetaSub,(Atom :- Body), Order),
     Order,
     substitue(metasub(Name, MetaSub), Prog1, Prog3),
     prove(Body, Prog3, Prog4),
     prove(As, Prog4, Prog2, Abds, Prob1, Prob2)
```

Figure 2: Prolog code for $Meta_{Abd}$.

3. According to Eq. 3 and 4, score each sampled $H \cup z$:

$$score(H, z) = P_{\sigma^*}(H|B)P_\theta(z|x) \tag{5}$$

4. Return the $H \cup z$ with the highest score.

3.3 The $Meta_{Abd}$ Implementation

Meta-Interpretive Learning [Muggleton *et al.*, 2015] is a form of ILP [Muggleton and de Raedt, 1994]. It learns first-order logic programs with a second-order meta-interpreter, which consists of a definite first-order background knowledge B and meta-rules M. B contains the primitive predicates for constructing first-order hypotheses H; M is second-order clauses with existentially quantified predicate variables and universally quantified first-order variables. In short, MIL attempts to prove the training examples and saves the resulting programs for successful proofs.

$Meta_{Abd}$ extends the general meta-interpreter of MIL by including an abduction procedure (bold fonts in Fig. 2) that can abduce groundings (e.g., specific constraints on pseudo-labels z). As shown in Fig. 2, it recursively proves a series of atomic goals by deduction (`deduce/1`), abducing explanatory facts (`call_abducible/3`) or generating a new clause from `meta-rule/4`.

The last argument of `call_abducible/3`, `Prob` $= P_\theta(z|x)$, describes the distribution of possible worlds collected from the raw inputs. It helps pruning the search space of the *abductive hypothesis* H. During the iterative refinement of H, $Meta_{Abd}$ greedily aborts its current `prove/6` procedure once it has a lower probability than the best abduction so far (the 8th line in Fig. 2).

After an abductive hypothesis H has been constructed, the search for z will be done by logical abduction. Finally, the score of $H \cup z$ will be calculated by Eq. 5, where $P_\theta(z|x)$ is the output of the perception model, which in this work is implemented with a neural network φ_θ that outputs:

$$P_\theta(z|x) = softmax(\varphi_\theta(x, z)).$$

Meanwhile, we define the prior distribution on H by following [Hocquette and Muggleton, 2018]:

$$P_{\sigma^*}(H|B) = \frac{6}{(\pi \cdot c(H))^2},$$

where $C(H)$ is the complexity of H, e.g., its size.

4 Experiments

This section describes the experiments of learning recursive arithmetic and sorting algorithms from images of handwritten digits[3], aiming to address the following questions: (1) Can $Meta_{Abd}$ learn first-order logic programs and train perceptual neural networks jointly? (2) Given the same or less amount of domain knowledge, is hybrid modelling, which directly leverages the background knowledge in symbolic form, better than end-to-end learning?

4.1 Cumulative Sum and Product From Images

Materials. We follow the settings in [Trask *et al.*, 2018]. The inputs of the two tasks are sequences of randomly chosen MNIST digits; the numerical outputs are the sum and product of the digits, respectively. The lengths of training sequences are 2–5. To verify if the learned models can extrapolate to longer inputs, the length of test examples ranges from 5 to 100. Note that for cumulative product, the extrapolation examples has maximum length 15 and only contain digits from 1 to 9. The dataset contains 3000 and 1000 examples for training and validation, respectively; the test data of each length has 10,000 examples.

Methods. We compare $Meta_{Abd}$ with following state-of-the-art baselines: End-to-end models include RNN, LSTM and LSTMs attached to Neural Accumulators(NAC) and Neural Arithmetic Logic Units (NALU) [Trask *et al.*, 2018]; NeSy system DeepProblog [Manhaeve *et al.*, 2018][4].

A convnet processes the input images to the recurrent networks and Problog programs, as [Trask *et al.*, 2018] and [Manhaeve *et al.*, 2018] described; it also serves as the perception model of $Meta_{Abd}$ to output the probabilistic facts. As shown in Tab. 1, NAC, NALU and $Meta_{Abd}$ are aware of the same amount of background knowledge for learning both perceptual convnet and recursive arithmetic algorithms jointly, while DeepProblog is provided with the ground-truth program and only trains the perceptual convnet.

Each experiment is carried out five times, and the average of the results are reported. The performance is measured by classification accuracy (Acc.) on length-one inputs, mean average error (MAE) in sum tasks, and mean average error on logarithm (log MAE) of the outputs in product tasks whose error grows exponentially with sequence length.

Results. Our experimental results are shown in Tab. 2; the learned first-order logic theories are shown in Fig. 3a. The end-to-end models that do not exploit any background knowledge (LSTM and RNN) perform worst. NALU and NAC is slightly better because they include neural cells with arithmetic modules, but the end-to-end learning pipeline based on embeddings results in low sample-efficiency. DeepProblog does not finish the training on the cumulative sum task and the test on cumulative product task within 72 hours because the recursive programs result in a huge groundings space for its maximum a posteriori (MAP) estimation.

[3]Code & data: https://github.com/AbductiveLearning/Meta_Abd

[4]We use NAC/NALU at https://github.com/kevinzakka/NALU-pytorch; DeepProblog at https://bitbucket.org/problog/deepproblog

Domain Knowledge	End-to-end Models	Neuro-Symbolic Models	Meta_{Abd}
Recurrence	LSTM & RNN	Problog's list operations	Prolog's list operations
Arithmetic functions	`NAC`& `NALU` [Trask *et al.*, 2018]	Full program of accumulative sum/product	Predicates `add`, `mult` and `eq`
Sequence & Odering	Permutation matrix $P_{\texttt{sort}}$ [Grover *et al.*, 2019]	Predicates ">", "=" and "<" [Dong *et al.*, 2019]	Prolog's `permutation`
Sorting	`sort` operator [Grover *et al.*, 2019]	`swap(i,j)` operator [Dong *et al.*, 2019]	Predicate `s` (learned from sub-task)

Table 1: Domain knowledge used by the compared models.

	MNIST cumulative sum				MNIST cumulative product			
	Acc.	MAE			Acc.	log MAE		
Sequence Length	1	5	10	100	1	5	10	15
LSTM	9.80%	15.3008	44.3082	449.8304	9.80%	11.1037	19.5594	21.6346
RNN-Relu	10.32%	12.3664	41.4368	446.9737	9.80%	10.7635	19.8029	21.8928
DeepProblog	Training timeout (72 hours)				**93.64%**	Test timeout (72 hours)		
LSTM-NAC	7.02%	6.0531	29.8749	435.4106	0.00%	9.6164	20.9943	17.9787
LSTM-NAC$_{10k}$	8.85%	1.9013	21.4870	424.2194	10.50%	9.3785	20.8712	17.2158
LSTM-NALU	0.00%	6.2233	32.7772	438.3457	0.00%	9.6154	20.9961	17.9487
LSTM-NALU$_{10k}$	0.00%	6.1041	31.2402	436.8040	0.00%	8.9741	20.9966	18.0257
$Meta_{Abd}$	**95.27%**	**0.5100**	**1.2994**	**6.5867**	**97.73%**	**0.3340**	**0.4951**	**2.3735**
LSTM-NAC$_{\text{1-shot CNN}}$	49.83%	0.8737	21.1724	426.0690	0.00%	6.0190	13.4729	17.9787
LSTM-NALU$_{\text{1-shot CNN}}$	0.00%	6.0070	30.2110	435.7494	0.00%	9.6176	20.9298	18.1792
$Meta_{Abd}$+1-shot CNN	**98.11%**	**0.2610**	**0.6813**	**4.7090**	**97.94%**	**0.3492**	**0.4920**	**2.4521**

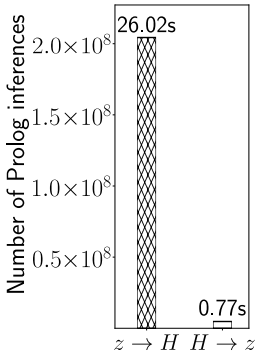
Table 2: Accuracy on the MNIST cumulative sum/product tasks.

Cumulative Sum:
```
f(A,B):-add(A,C),f(C,B).
f(A,B):-eq(A,B).
```

Cumulative Product:
```
f(A,B):-mult(A,C),f(C,B).
f(A,B):-eq(A,B).
```

Bogosort:
```
f(A,B):-permute(A,B,C),s(C).
s(A):-s_1(A,B),s(B).
s(A):-tail(A,B),empty(B).
s_1(A,B):-nn_pred(A),tail(A,B).
```

(a) Learned programs



(b) Abduction time.

Figure 3: Learned programs and the time efficiency of $Meta_{Abd}$.

The EM-based learning of $Meta_{Abd}$ may be trapped in local optima, which happens more frequently in cumulative sum than produce since its distribution $P(H, z|B, x, y, \theta)$ is much denser. Therefore, we also carry out experiments with one-shot pre-trained convnets, which are trained by randomly sampling one example in each class from MNIST data. Although the pre-trained convnet is weak at start (Acc. 20%∼35%), it provides a good initialisation and significantly improves the learning performance.

Fig. 3b compares the time efficiency between ILP's induction and $Meta_{Abd}$'s abduction-induction in one EM iteration of learning cumulative sum. "$z \to H$" means first sampling z and then inducing H with ILP; "$H \to z$" means first sampling an abductive hypothesis H and then using H to abduce z. The x-axis denotes the average number of Prolog inferences, the number at the end of each bar is the average inference time in seconds. Evidently, the abduction leads to a substantial improvement in the number of Prolog inferences and significantly the complexity of searching pseudo-labels.

4.2 Bogosort From Images

Materials. We follow the settings in [Grover *et al.*, 2019]. The input of this task is a sequence of randomly chosen MNIST images of distinct numbers; the output is the correct ranking (from large to small) of the digits. For example, when $x = [$$]$ then the output should be $y = [3, 1, 4, 5, 2]$ because the ground-truth labels $z^* = [5, 9, 4, 3, 8]$. The training dataset contains 3000 training/test and 1000 validation examples. The training examples are sequences of length 5, and we test the learned models on image sequences with lengths 3, 5 and 7.

Methods. We compare $Meta_{Abd}$ to an end-to-end model NeuralSort [Grover *et al.*, 2019] and a state-of-the-art NeSy approach Neural Logical Machines (NLM) [Dong *et al.*, 2019][5]. All experiments are repeated five times.

NeuralSort can be regarded as a differentiable approximation to bogosort (permutation sort). Given an input list of scalars, it generates a stochastic permutation matrix by applying the pre-defined deterministic or stochastic `sort` operator on the inputs. NLM can learn sorting through reinforcement learning in a domain whose states are described by vectors of relational features (groundings of dyadic predicates">", "==", "<") and action "`swap`". However, the original NLM only takes symbolic inputs[6], which provides a noisy-free relational features vector. In our experiments, we attach NLM with the same convnet as other methods to process images.

[5]We use NeuralSort at https://github.com/ermongroup/neuralsort; NLM at https://github.com/google/neural-logic-machines.

[6]Please see https://github.com/google/neural-logic-machines /blob/master/scripts/graph/learn_policy.py

Sequence Length	3	5	7
Neural Logical Machine (NLM)	17.97% (34.38%)	1.03% (20.27%)	0.01% (14.90%)
Deterministic NeuralSort	95.49% (96.82%)	88.26% (94.32%)	80.51% (92.38%)
Stochastic NeuralSort	95.37% (96.74%)	87.46% (94.03%)	78.50% (91.85%)
$Meta_{Abd}$	**96.33% (97.22%)**	**91.75% (95.24%)**	**87.42% (93.58%)**

Table 3: Accuracy of MNIST sort. First value is the rate of correct permutations; second value is the rate of correct individual element ranks.

We also compared to DeepProblog with the ground-truth program of sorting in this task, but it does not terminate when the neural predicate "`swap_net`"[7] is implemented to take noisy image inputs by the aforementioned convnet. Therefore, we do not display its performance in this task.

For $Meta_{Abd}$, it is easy to include stronger background knowledge for learning more efficient sorting algorithms [Cropper and Muggleton, 2019]. But in order to make a fair comparison to NeuralSort, we adapt the same background knowledge to logic program and let $Meta_{Abd}$ learn bogosort. The knowledge of permutation in $Meta_{Abd}$ is implemented with Prolog's built-in predicate `permutation`. Meanwhile, instead of providing the information about sorting as prior knowledge like the NeuralSort, we try to *learn* the concept of "sorted" (represented by a monadic predicate `s`) from data as a sub-task, whose training set is the subset of the sorted examples within the training dataset (< 20 examples). The two tasks are trained sequentially as a curriculum. $Meta_{Abd}$ learns the sub-task in the first five epochs and then re-uses the learned models to learn bogosort.

$Meta_{Abd}$ uses an MLP attached to the same untrained convnet as other models to produce dyadic probabilistic facts `nn_pred([`  `,`  `|_])`, which learns if the first two items in the image sequence satisfy a dyadic relation. Unlike NLM, the background knowledge of $Meta_{Abd}$ is agnostic to ordering, i.e., the dyadic `nn_pred` is not provided with supervision on whether it should learn "greater than" or "less than", so `nn_pred` only learns an unknown dyadic partial order among MNIST images. As we can see, the background knowledge used by $Meta_{Abd}$ is *much weaker* than the others.

Results. Tab. 3 shows the average accuracy of the compared methods in the sorting tasks; Fig. 3a shows the learned programs by $Meta_{Abd}$. The performance is measured by the average proportion of correct permutations and individual permutations following [Grover *et al.*, 2019]. Although using weaker background knowledge, $Meta_{Abd}$ has a significantly better performance than NeuralSort. Due to the high sample-complexity of reinforcement learning, NLM failed to learn any valid perceptual model and sorting algorithm (success trajectory rate 0.0% during training).

The learned program of `s` and the dyadic neural net `nn_pred` are both successfully re-used in the sorting task, where the learned program of `s` is consulted as interpreted background knowledge [Cropper *et al.*, 2020], and the neural network that generates probabilistic facts of `nn_pred` is directly re-used and continuously trained during the learning of sorting. This experiment also demonstrates $Meta_{Abd}$'s ability of learning recursive logic programs and predicate invention (the invented predicate `s_1` in Fig. 3a).

[7] Please see https://bitbucket.org/problog/deepproblog/src/master/examples/NIPS/Forth/Sort/quicksort.pl

5 Conclusion

In this paper, we present the Abductive Meta-Interpretive Learning ($Meta_{Abd}$) approach that can simultaneously train neural networks and learn recursive first-order logic theories with predicate invention. By combining ILP with neural networks, $Meta_{Abd}$ can learn human-interpretable logic programs directly from raw-data, and the learned neural models and logic theories can be directly re-used in subsequent learning tasks. $Meta_{Abd}$ adopts a general framework for combining perception with logical induction and abduction. The perception model extracts probabilistic facts from sub-symbolic data; the logical induction searches for first-order abductive theories in a relatively small hypothesis space; the logical abduction uses the abductive theory to prune the vast search space of the truth values of the probabilistic facts. The three parts are optimised together in a probabilistic model.

In future work, we would like to apply $Meta_{Abd}$ in real tasks such as computational science discovery, which is a typical abductive process that involve both symbolic domain knowledge and continuous/noisy raw data. Since $Meta_{Abd}$ uses pure logical inference for reasoning, it is possible to leverage more advanced symbolic inference/optimisation techniques like Satisfiability Modulo Theories (SMT) [Barrett and Tinelli, 2018] and Answer Set Programming (ASP) [Lifschitz, 2019] to reason more efficiently.

Acknowledgements

The first author acknowledges support from the UK's EPSRC Robot Synthetic Biologist, grant EP/R034915/1, for financial support. The second author acknowledges support from the UK's EPSRC Human-Like Computing Network, grant EP/R022291/1, for which he acts as director. The authors thank Céline Hocquette, Stassa Patsantzis and Ai Lun for their careful proofreading and helpful comments.

References

[Barrett and Tinelli, 2018] Clark W. Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[Bengio, 2017] Yoshua Bengio. The consciousness prior. *CoRR*, abs/1709.08568, 2017.

[Cohen *et al.*, 2020] William W. Cohen, Fan Yang, and Kathryn Mazaitis. Tensorlog: A probabilistic database implemented using deep-learning infrastructure. *Journal of Artificial Intelligence Research*, 67:285–325, 2020.

[Cropper and Muggleton, 2019] Andrew Cropper and Stephen H. Muggleton. Learning efficient logic programs. *Maching Learning*, 108(7):1063–1083, 2019.

[Cropper *et al.*, 2020] Andrew Cropper, Rolf Morel, and Stephen Muggleton. Learning higher-order logic programs. *Maching Learning*, 109(7):1289–1322, 2020.

[Dai and Zhou, 2017] Wang-Zhou Dai and Zhi-Hua Zhou. Combining logical abduction and statistical induction: Discovering written primitives with human knowledge. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 4392–4398, San Francisco, CA, 2017.

[Dai *et al.*, 2019] Wang-Zhou Dai, Qiu-Ling Xu, Yang Yu, and Zhi-Hua Zhou. Bridging machine learning and logical reasoning by abductive learning. In *Advances in Neural Information Processing Systems 32*, pages 2811–2822. Curran Associates, Inc., 2019.

[De Raedt *et al.*, 2020] Luc De Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. In Christian Bessiere, editor, *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pages 4943–4950. IJCAI, 7 2020.

[Dhingra *et al.*, 2020] Bhuwan Dhingra, Manzil Zaheer, Vidhisha Balachandran, Graham Neubig, Ruslan Salakhutdinov, and William W. Cohen. Differentiable reasoning over a virtual knowledge base. In *International Conference on Learning Representations*, Addis Ababa, Ethiopia, 2020. OpenReview.

[Donadello *et al.*, 2017] Ivan Donadello, Luciano Serafini, and Artur S. d'Avila Garcez. Logic tensor networks for semantic image interpretation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1596–1602, Melbourne, Australia, 2017. IJCAI.

[Dong *et al.*, 2019] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *International Conference on Learning Representations*, New Orleans, LA, 2019. OpenReview.

[Evans and Grefenstette, 2018] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[Evans *et al.*, 2021] Richard Evans, Matko Bošnjak, Lars Buesing, Kevin Ellis, David Pfau, Pushmeet Kohli, and Marek J. Sergot. Making sense of raw input. *Artificial Intelligence*, 299:103521, 2021.

[Flach *et al.*, 2000] Peter A. Flach, Antonis C. Kakas, and Antonis M. Hadjiantonis, editors. *Abduction and Induction: Essays on Their Relation and Integration*. Applied Logic Series. Springer Netherlands, 2000.

[Garcez *et al.*, 2019] Artur S. d'Avila Garcez, Marco Gori, Luís C. Lamb, Luciano Serafini, Michael Spranger, and Son N. Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *IfCoLog Journal of Logics and their Applications*, 6(4):611–632, 2019.

[Gaunt *et al.*, 2017] Alexander L. Gaunt, Marc Brockschmidt, Nate Kushman, and Daniel Tarlow. Differentiable programs with neural libraries. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1213–1222, Sydney, Australia, 2017. PMLR.

[Glasmachers, 2017] Tobias Glasmachers. Limits of end-to-end learning. In *Proceedings of The 9th Asian Conference on Machine Learning*, volume 77, pages 17–32, Seoul, Korea, 2017. PMLR.

[Grover *et al.*, 2019] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations*, New Orleans, LA, 2019. Openreview.

[Hocquette and Muggleton, 2018] Céline Hocquette and Stephen H. Muggleton. How much can experimental cost be reduced in active learning of agent strategies? In *Proceedings of the 28th International Conference on Inductive Logic Programming*, volume 11105, pages 38–53, Ferrara, Italy, 2018. Springer.

[Kahneman, 2011] Daniel Kahneman. *Thinking, fast and slow*. Farrar, Straus and Giroux, New York, 2011.

[Kakas *et al.*, 1992] Antonis C. Kakas, Robert A. Kowalski, and Francesca Toni. Abductive logic programming. *Journal of Logic Computation*, 2(6):719–770, 1992.

[Li *et al.*, 2020] Qing Li, Siyuan Huang, Yining Hong, Yixin Chen, Ying Nian Wu, and Song-Chun Zhu. Closed loop neural-symbolic learning via integrating neural perception, grammar parsing, and symbolic reasoning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 5884–5894, Online, 2020. PMLR.

[Lifschitz, 2019] Vladimir Lifschitz. *Answer Set Programming*. Springer, 2019.

[Manhaeve *et al.*, 2018] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems 31*, pages 3753–3763, Montréal, Canada, 2018. Curran Associates, Inc.

[Muggleton and de Raedt, 1994] Stephen H. Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19-20:629 – 679, 1994.

[Muggleton *et al.*, 2013] Stephen H. Muggleton, Dianhuan Lin, Jianzhong Chen, and Alireza Tamaddoni-Nezhad. MetaBayes: Bayesian meta-interpretative learning using higher-order stochastic refinement. In *Proceedings of the 23rd International Conference on Inductive Logic Programming*, volume 8812, pages 1–17, Rio de Janeiro, Brazil, 2013. Springer.

[Muggleton *et al.*, 2015] Stephen H. Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.

[Trask *et al.*, 2018] Andrew Trask, Felix Hill, Scott E Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems 31*, pages 8035–8044. Curran Associates, Inc., 2018.

[Wang *et al.*, 2019] Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6545–6554, Long Beach, CA, 2019. PMLR.

[Zhou *et al.*, 2018] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

[Zhou, 2019] Zhi-Hua Zhou. Abductive learning: towards bridging machine learning and logical reasoning. *Science China Information Sciences*, 62(7), 2019.

A Appendix

We introduce more implementation details and experimental results in the following sub-sections.

A.1 Parallel Abduction

As described in section 3.2, $Meta_{Abd}$ tries to estimate the most probable z by abduction following Eq. 3. Given training data $D = \{\langle x_i, y_i\rangle\}_{i=1}^{n}$, let $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{y} = (y_1, \ldots, y_n)$ and $\boldsymbol{z} = (z_1, \ldots, z_n)$, for $H \cup \boldsymbol{z}$ the posterior $P(H, \boldsymbol{z}|B, \boldsymbol{x}, \boldsymbol{y}, \theta) \propto P(H, \boldsymbol{y}, \boldsymbol{z}|B, \boldsymbol{x}, \theta)$, which can be further re-written as:

$$\begin{aligned} &P(\boldsymbol{y}|B, H, \boldsymbol{z})P_{\sigma^*}(H|B)P_\theta(\boldsymbol{z}|\boldsymbol{x}) \\ &\quad = P_{\sigma^*}(H|B)\prod_{i=1}^{n} P(y_i|B, H, z_i)P_\theta(z_i|x_i), \end{aligned} \tag{6}$$

where the last equation holds because the examples are drawn i.i.d. from the underlying distribution.

Therefore, the logical abduction in the expectation step of $Meta_{Abd}$ can be parallelised naturally:

1. Sample an abductive hypothesis H from the prior distribution $H \sim P_{\sigma^*}(H|B)$;
2. Parallelly abduce z_i from H and $\langle x_i, y_i\rangle$, and then calculate their scores by Eq. 5;
3. Aggregate the results by Eq. 6;
4. Get the best $H \cup \boldsymbol{z}$ and continue the maximisation step to optimise θ.

We applied this strategy in our implementation of $Meta_{Abd}$ and have achieved better efficiency on multi-threaded CPUs.

A.2 MNIST Cumulative Sum/Product

The background knowledge used in the MNIST cumulative sum/product experiments is shown in Fig. 4. We demonstrate how it works by the following example.

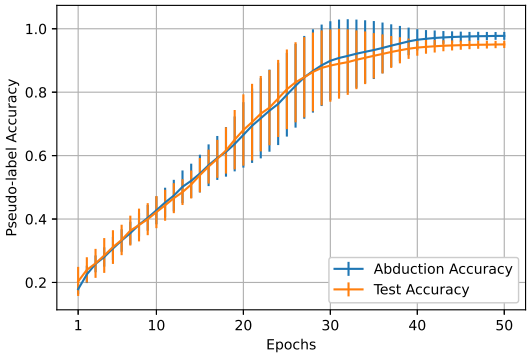
```
% Non-abducible primitives of list operations.
head([H|_],H).
tail([_|T],T).
empty([]).

% Abducible primitives for generating CLP constraints.
abduce_add([X,Y|T],[N|T],Abduced,1.0):-
     (not(ground(N)) ->
          metagol:new_var(N); number(N)),
     atomics_to_string([X,'+',Y,'#=',N], Abduced).
abduce_mult([X,Y|T],[N|T],Abduced,1.0):-
     (not(ground(N)) ->
          metagol:new_var(N); number(N)),
     atomics_to_string([X,'*',Y,'#=',N], Abduced).
abduce_eq([X|T],[N|T],Abduced,1.0):-
     (not(ground(N)) ->
          metagol:new_var(N); number(N)),
     atomics_to_string([X,'#=',N], Abduced).
```

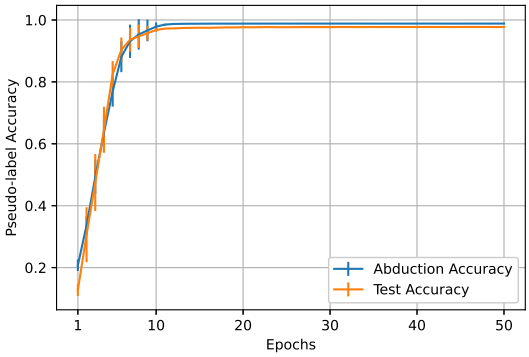
Figure 4: Background knowledge used in the MNIST cumulative sum/product tasks.

Example (Constraint abduction) Given a training example `f([`  `,`  `,`  `,`  `],15)`, $Meta_{Abd}$ will try to learn a program of the dyadic predicate `f` to satisfy (i.e., logically prove) the example. The program to be learned is the *abductive* hypothesis H. The learning process is similar to generic Meta-Interpretive Learning [Muggleton *et al.*, 2015] except that it abduces some ground expressions (the `Abduced` atom in Fig. 4) according to the definition of the abducible primitives. In the MNIST sum/product tasks, the `Abduced` atoms are strings like "`X+`  `#=3`", which is a CLP(Z)[8] constraint. According to the definition in Fig. 4, when the Prolog variable is not grounded (i.e., constant), the abducible variable will create a new variable to represent `N`; if the Prolog variable is grounded to a number, which means it is the final output in our example, then there is no need to generate a new variable to represent it. Assume that the currently sampled H is the cumulative sum program in Fig. 3a, then for the example `f([`  `,`  `,`  `,`  `],15)` $Meta_{Abd}$ can abduce four CLP(Z) constraints: "`+``#=N1`", "`N1+``#=N2`", "`N2+``#=N3`" and "`N3#=15`". Note that

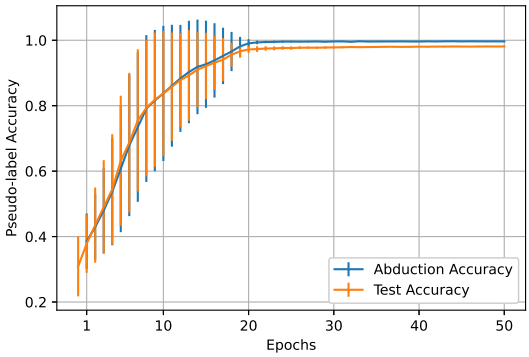
[8] https://github.com/triska/clpz



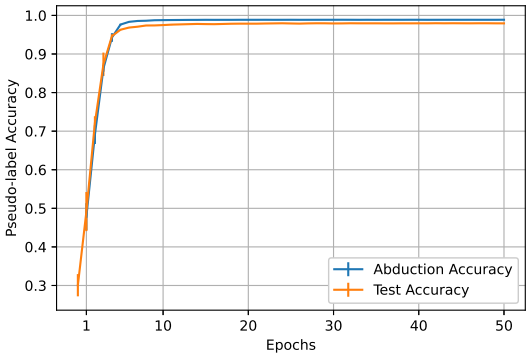
(a) MNIST sum



(b) MNIST product



(c) MNIST sum with 1-shot CNN pre-train



(d) MNIST product with 1-shot CNN pre-train

Figure 5: Pseudo-label accurracy during $Meta_{Abd}$ and $Meta_{Abd+\text{1-shot CNN}}$ learning.

```
nn(1,0,P00).  nn(1,1,P01).  nn(1,2,P02).  ...
nn(2,0,P10).  nn(2,1,P11).  nn(2,2,P12).  ...
nn(3,0,P20).  nn(3,1,P21).  nn(3,2,P22).  ...
nn(4,0,P30).  nn(4,1,P31).  nn(4,2,P32).  ...
```

Figure 6: Monadic probabilistic facts generated by neural network in the sum/product tasks.

the scores of the abducibles in Fig. 4 are all 1.0, which means that these constraints are *hard constraints* that have to be satisfied.

After abducing the constraints, $Meta_{Abd}$ will call the CLP(Z) to solve them, giving a small set of pseudo-labels z that satisfy those constraints. Then, $Meta_{Abd}$ will try to calculate the scores of the abduced $H \cup z$ according to Eq. 5. $P_{\sigma^*}(H|B)$ is directly given by H's complexity, i.e., the size of the program; $P_\theta(z|x)$ is given by the probabilistic facts by the perception neural network, which are shown in Fig. 6. The predicate "`nn(Img,Label,Prob)`" means the probability of `Img` being an instance of `Label` is `Prob`. To get the probability of all pseudo-labels of an image sequence, $Meta_{Abd}$ simply multiplies the probabilities of each image:

$$p_\theta(z|x) = \prod_j p_\theta(z_j|x_j),$$

where x_j is the j-th image in x (first argument of predicate `nn`), z_j is the abduced pseudo-label of x_j (second argument of `nn`), and the probability is the third argument of `nn`.

We also report the pseudo-label accuracy of abduction and perception during training, which are shown in Fig. 5. The blue lines are the accuracy of the abduced labels (i.e., the accuracy of the expectation of z) in each EM iteration; the orange lines are the accuracy of the perceptual neural net's classification accuracy on the MNIST test set. As we

```
% List operations.
head([H|_],H).
tail([_|T],T).
empty([]).

% Background knowledge about permutation
permute(L1,O,L2):-
    length(L1,N),
    findall(S,between(1,N,S),O1),
    % generate permutation with Prolog's built-in predicate
    catch(permutation(O1,O),_,fail),
    permute1(L1,O,L2).
% permute the image list with order O
permute1([],[],_).
permute1([S|List],[O|Os],List2):-
    nth1(O,List2,S),
    permute1(List,Os,List2).

% Abducible primitives.
abduce_nn_pred([X,Y|_],nn_pred(X,Y),Score):-
    nn_pred(X,Y,Score).
```

Figure 7: Background knowledge used in the MNIST sorting task.

```
nn_pred(X,Y,P) :- nn(X,Y,P), !.
nn_pred(X,Y,P) :- nn(Y,X,P1), P is 1-P1, !.

nn(1,2,P01).  nn(1,3,P02).  nn(1,9,P02).  ...
nn(2,3,P12).  nn(2,9,P13).  nn(3,9,P13).  ...
```

Figure 8: Dyadic probabilistic facts generated by neural network in the sorting task.

can observe, the convergence speed of cumulative sum is slower, because its the posterior distribution on pseudo-labels ($P(H, z|B, x, y, \theta)$) is much denser than that of cumulative product. After applying the one-shot CNN pre-train, whose test accuracy is shown at 0 epoch in the figures, the convergence speed of MNIST cumulative sum is significantly improved because the EM algorithm is less-likely to be trapped in local optimums.

A.3 MNIST Sorting

Different to the MNIST cumulative sum/product tasks which learn a perceptual neural network predicting the digit in each single image, in the MNIST sorting task, $Meta_{Abd}$ uses a perceptual neural network to learn an unknown binary relation between two images. Examples are shown in Fig. 8. The neural network uses the same convnet as before to take the input from a pair of images (the first two arguments of predicate `nn`), and then a Multi-Layered Perception (MLP) is used to predict the probability `PIJ`. The first two clauses translate the neural network's output `nn` to the probabilistic facts for $Meta_{Abd}$'s abduction.

Example (Dyadic facts abduction) Background knowledge of the MNIST sorting task is shown in Fig. 7. Different to the previous example which abduces the label of each input image, in the sorting task, the facts being extracted from raw data are dyadic relationship between two images. Given an training example with input $x =$ [1,2,3,9], the perceptual neural network will process all the pairwise combinations among them and output a score as shown in Fig. 8. Because the pairwise combinations are just a half of pairwise permutations, we also provided a symmetric rule to complete them (the first two clauses in Fig. 8). During $Meta_{Abd}$'s induction, the abduced facts are the pairwise probabilistic facts themselves instead of CLP(Z) constraints like before, so the `Score` is the probability of each probabilistic fact. In other words, in the sorting task, *the abduction of* z (the truth values of the probabilistic facts) *is performed simultaneously with logical induction*. Recall the Prolog code of $Meta_{Abd}$ in Fig. 2, there is a greedy process that keeps the current most probable abduction with `getmaxprob(Max)` and `setmaxprob(Max)`. The greedy strategy is used to prune the search space of z, it excludes the facts with low probability and quickly find a locally optimal z (truth value assignment), which will be used as pseudo-labels to train the perceptual neural network in the maximisation step.

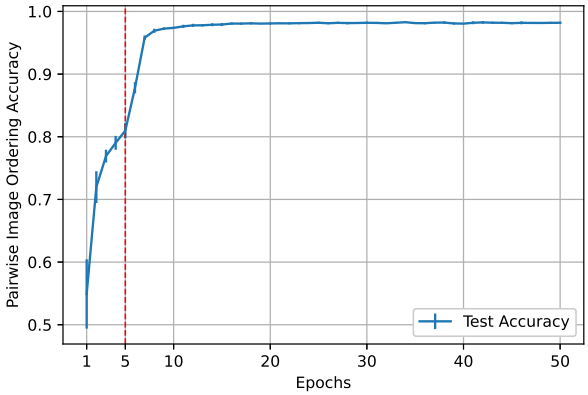


Figure 9: MNIST pairwise ordering (`nn_pred`) accuracy during learning.

```
metarule([P,Q],[P,A],[[Q,A]]).
metarule([P,Q],[P,A],[[Q,A,B],[P,B]]).
metarule([P,Q,R],[P,A],[[Q,A,B],[R,B]]).
metarule([P,Q,R],[P,A,B],[[Q,A],[R,A,B]]).
metarule([P,Q],[P,A,B],[[Q,A,B]]).
metarule([P,Q,R],[P,A,B],[[Q,A,B],[R,A,B]]).
metarule([P,Q,R],[P,A,B],[[Q,A,B,C],[R,C]]).
metarule([P,Q,R],[P,A,B],[[Q,A,B],[R,B]]).
metarule([P,Q,R],[P,A,B],[[Q,A,C],[R,C,B]]).
```

Figure 10: Meta-rules used in all the experiments.

Fig. 9 shows the perception accuracy during training. The test pairs contains 10,000 randomly sampled images from the MNIST test set. The vertical line at epoch 5 shows the time point when $Meta_{Abd}$ switching from the sub-task (learning concept of "sorted" with target predicate `s`) to the main tasks (learning permutation sort). The results in this figure verifies that the perception model is successfully re-used in this experiment.

A.4 Reproducibility

We introduce more experimental details in this subsection. All experiments are completed on a PC with AMD Ryzen 3900X CPU and Nvidia 2080Ti GPU. The data and source codes of $Meta_{Abd}$ will be available after the publication of this work.

meta-rules

The meta-interpreter of $Meta_{Abd}$ uses a set of meta-rules to guide the induction of the logic theory H. We use the meta-rules from the higher-order meta-interpreter $Metagol_{ho}$[9] [Cropper *et al.*, 2020], which are shown in Fig. 10.It has been shown that these meta-rules have universal Turing expressivity and can represent higher-order programs [Cropper *et al.*, 2020].

We further compared the inference speed of $Meta_{Abd}$ with different sizes of meta-rules. Specifically, following are the time difference measured by the average number of Prolog inferences in each batch of 's abduction-induction inference in the accumulative sum task. The settings are as follows:

- $Meta_{Abd}$ contains at least one metarule, which is `P(A,B):-Q(A,B)`, i.e., calling a primitive function. However, it is not complete for representing the hypothesis space since none of the primitive predicates is able to define the target concept (otherwise they won't be called as "primitives"). Hence, we start from at least 2 meta-rules;
- The perceptual CNN is randomly initialised and un-trained, i.e., the distribution of probabilistic facts is random, which is the *worst-case* for abduction, so the result here is slower than the average result in Fig. 3b;
- Choosing metarules is a subset selection problem. Following the traditions in combinatorial optimisation, we report the worst result among all varied combinations;

[9] https://github.com/andrewcropper/mlj19-metaho

Number of meta-rules	Number of Prolog inferences	Time (seconds)
9	26324856	1.571
8	26324638	1.567
7	26324626	1.567
6	26324287	1.527
5	26324009	1.528
4	26321479	1.521
3	26314047	1.521
2	10991735	0.635

Table 4: Time costs (worst-case) of using different numbers of meta-rules. Note that the setting of using only 2 meta-rules is equivalent to RNNs which are forced to learn a minimum recursive program.

- The number of Prolog inferences includes the CLP(Z) optimisation.

As we can see from Fig. 4, there is not much difference between using 9–3 metarules for $Meta_{Abd}$ (when the program hypothesis space is complete). Hence, if the users have a strong bias on the target theory and only use the relevant metarules, the search speed can be very fast.

Neural Network & Hyperparameters

The convnet in our experiments is from PyTorch's MNIST tutorial[10] as [Trask *et al.*, 2018] suggested. The LSTM and RNN models in the MNIST cumulative sum/product experiments have 2 hidden layers with dimension 64; the `NAC` and `NALU` modules have 2 hidden layers with dimension 32. In the MNIST sorting experiments, we set the hyperparameter $\tau = 1.0$ for NeuralSort, which is the default value in the original codes[11]. Moreover, the output of NeuralSort is a vector with floating numbers, in order to reproduce the result from the original paper, we rank the output scores to generate the final prediction of orderings.

DeepProblog [Manhaeve *et al.*, 2018] and Neural Logical Machines (NLM) [Dong *et al.*, 2019] are treated as blackbox models attached with the same convnet as $Meta_{Abd}$. The Problog programs of DeepProblog are the ground-truth programs in Fig. 3a; the parameters of NLM is tuned following the instructions in its repository.

[10] https://github.com/pytorch/examples/tree/master/mnist

[11] https://github.com/ermongroup/neuralsort