Bridging Machine Learning and Logical Reasoning by Abductive Learning*

Wang-Zhou Dai† **Qiuling Xu**† **Yang Yu**† **Zhi-Hua Zhou**

National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
{daiwz, xuql, yuy, zhouzh}@lamda.nju.edu.cn

Abstract

Perception and reasoning are two representative abilities of intelligence that are integrated seamlessly during human problem-solving processes. In the area of artificial intelligence (AI), the two abilities are usually realised by machine learning and logic programming, respectively. However, the two categories of techniques were developed separately throughout most of the history of AI. In this paper, we present the *abductive learning* targeted at unifying the two AI paradigms in a mutually beneficial way, where the machine learning model learns to perceive primitive logic facts from data, while logical reasoning can exploit symbolic domain knowledge and correct the wrongly perceived facts for improving the machine learning models. Furthermore, we propose a novel approach to optimise the machine learning model and the logical reasoning model jointly. We demonstrate that by using *abductive learning*, machines can learn to recognise numbers and resolve unknown mathematical operations simultaneously from images of simple hand-written equations. Moreover, the learned models can be generalised to longer equations and adapted to different tasks, which is beyond the capability of state-of-the-art deep learning models.

1 Introduction

Human cognition [34] consists of two remarkable capabilities: perception and reasoning, where the former one processes sensory information, and the latter one majorly works symbolically. These two abilities function at the same time and affect each other, and they are often joined subconsciously by humans, which is essential in many real-life learning and problem-solving procedures [34].

Modern artificial intelligence (AI) systems exhibit both these abilities. Machine learning techniques such as deep neural networks have achieved extraordinary performance in solving perception tasks [19]; meanwhile, logic-based AI systems have succeeded in human-level reasoning abilities in proving mathematical theorems [27] and in performing inductive reasoning concerning relations [25].

However, popular machine learning techniques can hardly exploit sophisticated domain knowledge in symbolic forms, and perceived information is hard to include in reasoning systems. Even in recent neural networks with the ability to focus on relations [31], enhanced memories and differentiable knowledge representations [13], full logical reasoning ability is still missing—as an example, consider the difficulties of understanding natural language [17]. On the other hand, Probabilistic Logic Program (PLP) [5] and Statistical Relational Learning (SRL) [12] are aiming at integrating learning and logical reasoning by preserving the symbolic representation. However, they usually require semantic-level input, which involves pre-processing sub-symbolic data into logic facts [30].

†These authors contributed equally to this work.

*W.-Z. Dai (w.dai@imperial.ac.uk) and Q. Xu (simpleword2014@gmail.com) are now at Imperial College London and Purdue University, respectively.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

To leverage learning and reasoning more naturally, it is crucial to understand how perception and reasoning affect each other in a single system. A possible answer is *abduction* [28], or termed as *retro-production* [33]. It refers to the process of selectively inferring specific facts and hypotheses that give the best explaination to observations based on background knowledge [23, 14], where the "observations" are mostly sensory information, and "knowledge" is usually symbolic and structural.

An example of human abductive problem-solving is the decipherment of Mayan hieroglyphs [15], which reflects two remarkable human intelligence capabilities: 1) visually perceiving individual numbers from hieroglyphs and 2) reasoning symbolically based on the background knowledge about mathematics and calendars. Fig. 1 shows a Mayan calendar discovered from the Palenque Temple of the Cross Complex, it starts with the mythical *creation date*, followed by a time period written in *long count*, and finished with a specific date encoded by *Tzolk'in* and *Haab'* calendars. Fig. 2 depicts the records of breaking Fig. 1 by Charles P. Bowditch [2]. He first identified some known numbers, and confirmed that the first and sixth hieroglyphs are the same. Then, Bowditch tried substituting those unknown hieroglyphs with visually similar numbers, as shown in "Column 1" in Fig. 2. Meanwhile, he calculated the *Tzolk'in* and *Haab'* values according to his conjectures and background knowledge in Mayan calendars, as shown in "Column 2" in Fig. 2. Finally, he got the correct answer "1.18.5.4.0, 1 Ahau 13 Mak" by observing the *consistency* between his conjecture and calculation [2].

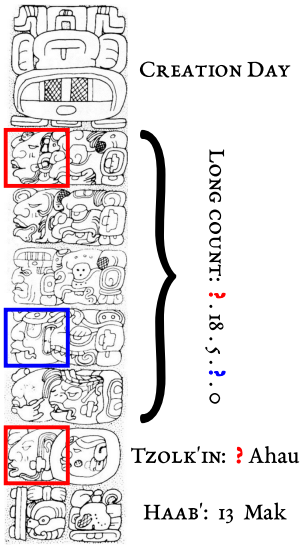


Figure 1: A Mayan calendar. The coloured boxes and "?" correspond to unknown numbers.

Inspired by abductive problem-solving, we present the *Abductive Learning* (ABL), a new approach towards bridging machine learning and logical reasoning. In *abductive learning*, a machine learning model is responsible for interpreting sub-symbolic data into primitive logical facts, and a logical model can reason about the interpreted facts based on some first-order logical background knowledge to obtain the final output. The primary difficulty lies in the fact that the sub-symbolic and symbolic models can hardly be trained together. More concretely: 1) it does not have any ground-truth of the primitive logic facts — e.g., the correct numbers in Fig. 1 — for training the machine learning model; 2) without accurate primitive logic facts, the reasoning model can hardly deduce the correct output or learn the right logical theory.

Column 1.			Column 2.
9	18.5.0.0.,	9 Ahau 13 Mac.	4 Ahau 13 Ceh(18)
9	18.5.1.0.,	9 Ahau 13 Mac.	11 Ahau 13 Mac(18)
9	18.5.2.0.,	9 Ahau 13 Mac.	5 Ahau 13 Kankin(18)
9	18.5.3.0.,	9 Ahau 13 Mac.	12 Ahau 13 Muan(18)
9	18.5.4.0.,	9 Ahau 13 Mac.	6 Ahau 13 Pax(18)
9	18.5.5.0.,	9 Ahau 13 Mac.	13 Ahau 13 Kayab(18)
8	18.5. 8.0.,	8 Ahau 13 Mac	9 Ahau 3 Zac(40)
8	18.5. 9.0.,	8 Ahau 13 Mac	3 Ahau 3 Ceh(40)
8	18.5.10.0.,	8 Ahau 13 Mac	10 Ahau 3 Mac(40)
8	18.5.11.0.,	8 Ahau 13 Mac	4 Ahau 3 Kankin(40)
1	18.5.2.0.,	1 Ahau 13 Mac	13 Ahau 13 Zac(34)
1	18.5.3.0.,	1 Ahau 13 Mac	7 Ahau 13 Ceh(34)
1	18.5.4.0.,	1 Ahau 13 Mac	1 Ahau 13 Mac(34)
1	18.5.5.0.,	1 Ahau 13 Mac	8 Ahau 13 Kankin(34)
1	18.5.6.0.,	1 Ahau 13 Mac	2 Ahau 13 Muan(34)
1	18.5.7.0.,	1 Ahau 13 Mac	9 Ahau 13 Pax(34)

Figure 2: Bowditch's decipherment of Fig. 1 (he wrote "Mak" as "Mac") [2]. Numbers in the vertical boxes are his guesses (Column 1) to the unknown hieroglyphs in Fig. 1. The dashed yellow box marks the consistent result according to his calculation (Column 2).

Our presented *Abductive Learning* (ABL) tries to address these challenges with logical abduction [18, 7] and consistency optimisation. Given a training sample associated with a final output, logical abduction can conjecture about the missing information — e.g., candidate primitive facts in the example, or logic clauses that can complete the background knowledge — to establish a consistent proof from the sample to its final output. The abduced primitive facts and logic clauses are then used for training the machine learning model and stored as symbolic knowledge, respectively. Consistency optimisation is used for maximising the consistency between the conjectures and the background knowledge. To solve this highly complex problem, we transform it into a task that searches for a function guessing about possibly mistaken primitive facts.

Because of the difficulty of collecting Mayan hieroglyph data, we designed a similar task — the handwritten equation decipherment puzzles — for experiments. The task is to learn image recognition (perception) and mathematical operations for calculating the equations (reasoning) simultaneously. Experimental results show that ABL generalise better than state-of-the-art deep learning models and can leverage learning and reasoning in a mutually beneficial way. Further experiments on a visual n-queens task shows that the ABL framework is flexible and can improve the performance of machine learning by taking advantage of classical symbolic AI systems such as Constraint Logic Programming [16].

2 Related Work

As one of the holy grail problems in AI, combining machine learning and logical reasoning has drawn much attention. Most existing methods try to combine the two different systems by making one side to subsume the other. For example, Fuzzy logic [41], Probabilistic Logic Programming [5] and Statistical Relational Learning [12] have been presented to empower traditional logic-based methods to handle uncertainty; however, most of them still require human-defined symbols as input [30]. Probabilistic programming [35, 21, 20] is presented as an analogy to human cognition to enable probabilistic reasoning with sub-symbolic primitives, yet the correspondence between the sub-symbolic primitives and their symbolic representations used in programming is assumed to already exist rather than assuming that it should be learned.

Another typical approach is to use deep neural networks or other differentiable functional calculations to approximate symbolic calculi. Some of them try to translate logical programs into neural networks, e.g. KBANN [38] and Artur Garcez's works on neural-symbolic learning [10, 9]; others directly replace symbolic computing with differentiable functions, e.g., differential programming methods such as DNC and so on attempt to emulate symbolic computing using differentiable functional calculations [13, 11, 1, 6]. However, few of them can make full-featured logical inferences, and they usually require large amounts of training data.

Different from the previous works, ABL tries to bridge machine learning and logical reasoning in a mutually beneficial way [42]. The two components perceive sub-symbolic information and make symbolic reasoning separately but interactively. The logical abduction with consistency optimisation enables ABL to improve the machine learning model and learn logical theory in a single framework.

3 Abductive Learning

In this section, we present the ABL approach. Notations and the problem formulation are firstly introduced, followed by the detailed description and the presented optimisation approach.

3.1 Problem Setting

The task of *abductive learning* can be formalised as follows. The input of *abductive learning* consists of a set of labelled training data $D = \{\langle \boldsymbol{x}_1, y_1 \rangle, \ldots, \langle \boldsymbol{x}_n, y_n \rangle\}$ about a target concept C and a domain knowledge base B, where $\boldsymbol{x}_i \in \mathcal{X}$ is the input data, $y_i \in \{0, 1\}$ is the label for $\boldsymbol{x}_i$ of target concept C, and B is a set of first-order logical clauses. The target concept C is defined with unknown relationships amongst a set of primitive concepts symbols $\mathcal{P} = \{p_1, \ldots, p_r\}$ in the domain, where each p_k is a defined symbol in B. The target of abductive learning is to output a hypothesis model $H = p \cup \Delta_C$, in which:

- $p : \mathcal{X} \mapsto \mathcal{P}$ is a mapping from the feature space to primitive symbols, i.e., it is a *perception model* formulated as a conventional machine learning model;
- Δ_C is a set of first-order logical clauses that define the target concept C with B, which is called *knowledge model*.

The hypothesis model should satisfy:

$$\forall \langle \boldsymbol{x}, y \rangle \in D \, (B \cup \Delta_c \cup p(\boldsymbol{x}) \models y). \tag{1}$$

Where "$\models$" stands for logical entailment.

As we can observe from Eq. 1, the major challenge for abductive learning is that the perception model p and the knowledge model Δ_C are *mutually dependent*: 1) To learn Δ_C, the perception results $p(\boldsymbol{x})$ — the set of groundings of the primitive concepts in $\boldsymbol{x}$ — is required; 2) To obtain p, we need to get the ground truth labels $p(\boldsymbol{x})$ for training, which can only be logically derived from $B \cup \Delta_C$ and y. When the machine learning model is under-trained, the perceived primitive symbols $p(\boldsymbol{x})$ is highly possible to be incorrect; therefore, we name them *pseudo-groundings* or *pseudo-labels*. As a consequence, the inference of Δ_C based on Eq. 1 would be inconsistent; when the knowledge model Δ_C is inaccurate, the logically derived pseudo-labels $p(\boldsymbol{x})$ might also be wrong, which harms the training of p. In either way, they will interrupt the learning process.

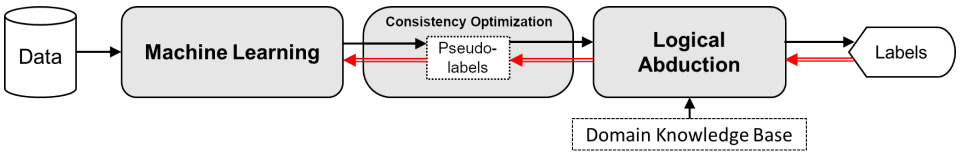


Figure 3: The structure of ABL framework.

3.2 Framework

The ABL framework [42] tries to address these challenges by connecting machine learning with an abductive logical reasoning module and bridging them with consistency optimisation. Fig. 3 shows the outline of the framework.

Machine learning is used for learning the perception model p. Given an input instance $\boldsymbol{x}$, p can predict the pseudo-labels $p(\boldsymbol{x})$ as groundings of possible primitive concepts in $\boldsymbol{x}$. When the pseudo-labels contain mistakes, the perception model needs to be re-trained, where the labels are the revised pseudo-labels $r(\boldsymbol{x})$ returned from logical abduction.

Logical abduction is the logical formalisation of abductive reasoning. Given observed facts and background knowledge expressed as first-order logical clauses, logical abduction can abduce ground hypotheses as possible explanations to the observed facts. A declarative framework in Logic Programming that formalises this process is Abductive Logic Programming (ALP) [18]. Formally, an abductive logic program can be defined as follows:

Definition 1 [18] *An abductive logic program is a triplet* (B, A, IC)*, where* B *is* background knowledge*,* A *is a set of* abducible predicates*, and* IC *is the* integrity constraints*. Given some* observed facts O*, the program outputs a set* Δ*, of ground abducibles of* A*, such that:*

- $B \cup \Delta \models O$*,*
- $B \cup \Delta \models IC$*,*
- $B \cup \Delta$ *is consistent.*

Intuitively, the abductive explanation Δ serves as a hypothesis that explains how an observation O could hold according to the background knowledge B and the constraint IC.

Considering the formulation in Eq. 1, ABL takes the instance labels about the final concept as observed facts, and takes the hypothesis model $H = \Delta_C \cup p$ as abducibles. Given a fixed Δ_C, ABL can abduce $p(X)$ according to B and Y; when the perception model p has been determined, ALP is able to abduce the knowledge model Δ_C according to $B \cup p(X) \cup Y$. Here we use $p(X) = \bigcup_{i=1}^{n} \{p(\boldsymbol{x}_i)\}$ to represent the pseudo-labels of all the instances $X = \bigcup_{i=1}^{n} \{\boldsymbol{x}_i\}$, and $Y = \bigcup_{i=1}^{n} \{y_i\}$ are the final concept labels corresponding to X. Therefore, we can denote the abduced knowledge model conditioned by $B \cup p(X)$ and Y as $\Delta_C(B \cup p(X), Y)$.

3.3 Optimisation

The objective of ABL is to learn a hypothesis consistent with background knowledge and training examples. More concretely, ABL tries to maximise the consistency between the abduced hypotheses H with training data $D = \{\langle \boldsymbol{x}_i, y_i \rangle\}_{i=1}^{n}$ given background knowledge B:

$$\max_{H = p \cup \Delta_C} \quad \mathrm{Con}(H \cup D; B), \tag{2}$$

where $\mathrm{Con}(H \cup D; B)$ stands for the size of subset $\hat{D}_C \subseteq D$ which is consistent with $H = p \cup \Delta_C$ given B. It can be defined as follows:

$$\begin{aligned} \mathrm{Con}(H \cup D; B) = \max_{D_c \subseteq D} \quad & | \, D_c \, | \\ \text{s.t.} \quad & \forall \langle \boldsymbol{x}_i, y_i \rangle \in D_c \ (B \cup \Delta_C \cup p(\boldsymbol{x}_i) \models y_i) \, . \end{aligned} \tag{3}$$

To solve Eq. 2, ABL tries to optimise Δ_C and p alternatively.

During the t-th epoch, when the perception model p^t is under-trained, the pseudo-labels $p^t(X)$ could be incorrect and make logical abduction fail to abduce any consistent Δ_C satisfying Eq. 1, resulting in $\mathrm{Con}(H \cup D; B) = 0$.

Therefore, ABL needs to correct the wrongly perceived pseudo-labels to achieve consistent abductions, such that Δ_C^t can be consistent with as many as possible examples in D. Here we denote the pseudo-labels to be revised as $\delta[p^t(X)] \subseteq p^t(X)$, where δ is a heuristic function to estimate which pseudo-labels are perceived incorrectly by current machine learning model p^t — in analogy to Bowditch's power of identifying the misinterpreted hieroglyphs (see Fig. 2).

After removing the incorrect pseudo-labels marked by the δ function, ABL can apply logical abduction to abduce the candidate pseudo-labels to revise $\delta[p^t(X)]$ together with Δ_C^t by considering:

$$B \cup p^t(X) - \delta[p^t(X)] \cup \Delta_{\delta[p^t(X)]} \cup \Delta_C^t \models Y \tag{4}$$

Where $p^t(X) - \delta[p^t(X)]$ are the remaining "correct" pseudo-labels determined by δ, and $\Delta_{\delta[p^t(X)]}$ are the abduced pseudo-labels for revising $\delta[p^t(X)]$.

Theoretically, δ can simply mark all pseudo-labels as "wrong", i.e., letting $\delta[p^t(X)] = p^t(X)$ and ask logical abduction to do all the learning jobs. In this case, ABL can always abduce a consistent $\Delta_{\delta[p^t(X)]} \cup \Delta_C^t$ satisfying Eq. 4. However, this means that the logical abduction have to learn the knowledge model Δ_C without any influence from the perception model p and the raw data X. It not only results in an exponentially larger search space for the abduction, but also breaks the link between logical reasoning and actual data. Consequently, ABL chooses to restrict the revision to be not too far away from the percieved results, by limiting $\mid \delta[p^t(X)] \mid \leq M$, where M defines the step-wise search space on the scale of the abduction and is sufficient to be set a small number.

Therefore, when p^t is fixed, we can transform the optimisation problem of Δ_C into an optimisation problem of function δ, and reformulate Eq. 2 as follows:

$$\begin{aligned} \max_{\delta} \quad & \mathrm{Con}(H_\delta \cup D), \\ s.t. \quad & \mid \delta[p^t(X)] \mid \leq M \end{aligned} \tag{5}$$

where $H_\delta = p^t(X) - \delta[p^t(X)] \cup \Delta_{\delta[p^t(X)]} \cup \Delta_C^t$ is the abduced hypothesis defined by Eq. 4. Although this objective is still non-convex, optimising δ instead of Δ_C allows ABL to revise and improve the hypothesis even when p^t is not optimal.

The heuristic function δ could take any form as long as it can be easily learned. We present to solve it with derivative-free optimisation [40], which is a flexible framework for optimising non-convex objectives. As to the subset selection problem in Eq. 5, we present to solve it with greedy algorithms.

After obtained the δ and Δ_C^t, ABL can directly apply logical abduction to obtain the revised pseudo-labels $r(X) = p^t(X) - \delta[p^t(X)] \cup \Delta_{\delta[p^t(X)]}$, which is used for re-training the machine learning model. This procedure can be formulated as follows:

$$p^{t+1} = \arg\min_{p} \quad \sum_{i=1}^{m} \mathrm{Loss}\left(p(\boldsymbol{x}_i), r(\boldsymbol{x}_i)\right), \tag{6}$$

where Loss stands for the loss function for machine learning, $r(\boldsymbol{x}_i) \in r(X)$ is the set of revised pseudo-labels for instance $\boldsymbol{x}_i \in X$.

In short, ABL works as follows: Given the training data, an initialised machine learning model is used for obtaining the pseudo-labels, which are then treated as groundings of the primitive concepts for logical reasoning to abduce Δ_C. If the abduction terminated due to inconsistency, the consistency optimisation procedure in Eq. 5 is called to revise the pseudo-labels, which are then used for re-training the machine learning model.

4 Implementation

To verify the effectiveness of the presented approach, we designed the handwritten equation decipherment tasks, as shown in Fig. 4 and applied ABL to solve them.

The equations for the decipherment tasks consist of sequential pictures of characters. The equations are constructed from images of symbols ("0", "1", "+" and "="), and they are generated with *unknown* operation rules, each example is associated with a label that indicates whether the equation is correct. A machine is tasked with learning from a training set of labelled equations, and the trained model is

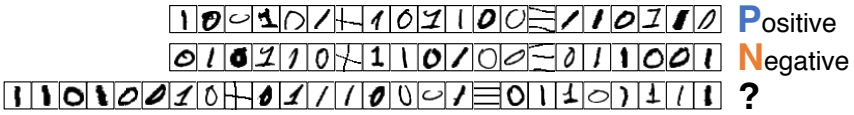


Figure 4: Handwritten equation decipherment puzzle: a computer should learn to recognise the symbols and figure out the unknown operation rules ("xnor" in this example) simultaneously.

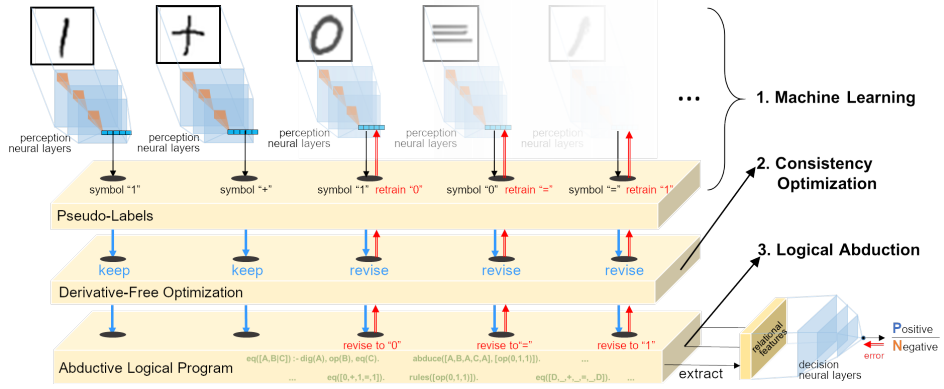


Figure 5: The structure of our ABL implementation.

expected to predict unseen equations correctly. Thus, this task demands the same ability as a human jointly utilising perceptual and reasoning abilities in Fig. 1.

Fig. 5 shows the architecture of our ABL implementation, which employs a convolutional neural network (CNN) [22] as the perception machine learning model. The CNN takes image pixels as input and is expected to output the symbols in the image. The symbol output forms the pseudo-labels. The logical abduction is realised by an Abductive Logic Program implemented with Prolog. The consistency optimisation problem in Eq. 5 is solved by a derivative-free optimisation tool RACOS[40].

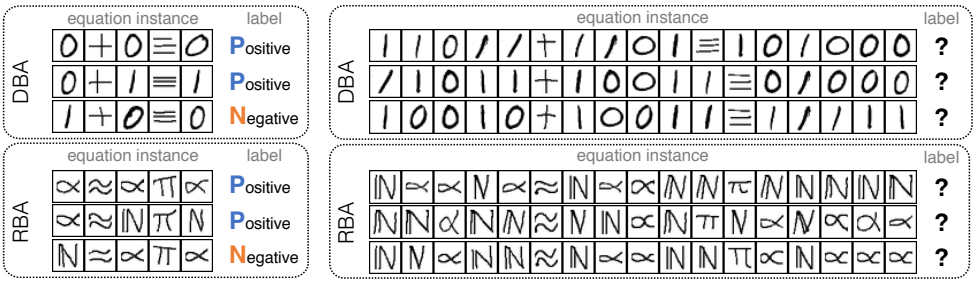
Before training, the domain knowledge—written as a logic program—is provided to the ALP as background knowledge B. In our implementation, B involves only the *structure* of the equations and a *recursive* definition of *bit-wise operations*. The background knowledge about equation *structures* is a set of definite clause grammar (DCG) rules recursively define that a digit is a sequence of "0" and "1", and each equation share the structure of `X+Y=Z`, although the length of `X`, `Y` and `Z` may be varied. The knowledge about *bit-wise operations* is a recursive logic program that reversely calculate `X+Y`, i.e., it operates on `X` and `Y` digit-by-digit and from the last digit to the first. The logic programs defining this background knowledge are shown in the supplementary.

Remark Please notice that, the specific rules for calculating the operations are *undefined* in B, i.e., results of "0+0", "0+1" and "1+1" could be "0", "1", "00", "01" or even "10". The missing calculation rules form the knowledge model Δ_C, which are required to be *learned* from the data.

After training starts, the CNN will interpret the images to the symbolic equations constructed by pseudo-labels "0", "1", "+" and "=". Because the CNN is untrained, the perceived symbols are typically wrong. In this case, ALP cannot abduce any Δ_C that is consistent with the training data according to the domain knowledge, i.e., no calculation rules can satisfy the perceived pseudo-labels with the associated labels. To abduce the most consistent Δ_C, ABL learns the heuristic function δ for marking possible incorrect pseudo-labels.

For example, in the beginning, the under-trained CNN is highly likely to interpret the images as a pseudo-grounding $\texttt{eq}_0$`=[1,1,1,1,1]`, which is inconsistent with any binary operations since it has no operator symbol. Observing that ALP cannot abduce a consistent hypothesis, RACOS will learn a δ that substituting the "possibly incorrect" pseudo-labels in $\texttt{eq}_0$ with blank Prolog variables, e.g., $\texttt{eq}_1$`=[1,_,1,_,1]`. Then, ALP can abduce a consistent hypothesis involving the operation rule `op(1,1,[1])` and a list of revised pseudo-labels $\texttt{eq}_1$`'=[1,+,1,=,1]`, where the latter one is used for re-train the CNN, helping it distinguish images of "+" and "=" from other symbols.

The complexity of the optimisation objective in Eq. 5 is very high, which usually makes it infeasible to evaluate the entire training set D during optimisation. Therefore, ABL performs abduction and optimisation for T times, each time using a subsample $D_t \subseteq D$ for training. The locally consistent reasoning model Δ_C^t abduced in each iteration are kept as a relational feature.



(a) Training examples. (b) Test examples.

Figure 6: Data examples for the handwritten equations decipherment tasks.

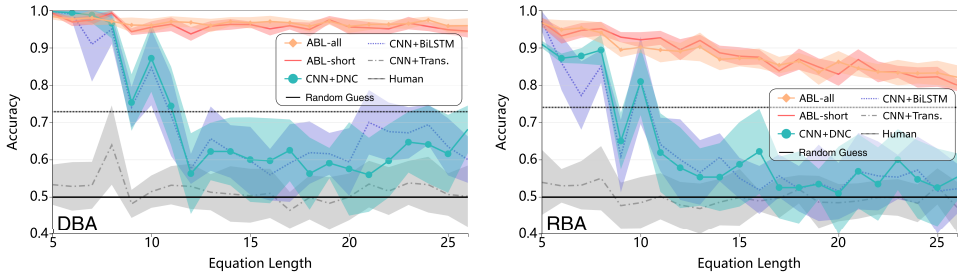


Figure 7: Experimental results of the DBA (left) and RBA (right) tasks.

After the CNN converged or the algorithm meets the iteration limit, all $\langle \boldsymbol{x}_i, y_i \rangle \in D$ are propositionalised to binary feature vectors by the relational features. For every input equation $\boldsymbol{x}_i$, its pseudo-labels will be evaluated by all the relational features to produce a binary vector $\boldsymbol{u}_i = [u_{i1}, \ldots, u_{iT}]$, where

$$u_{ij} = \begin{cases} 1, & B \cup \Delta_C^j \cup p(\boldsymbol{x}_i) \models y_i, \\ 0, & B \cup \Delta_C^j \cup p(\boldsymbol{x}_i) \not\models y_i. \end{cases} \tag{7}$$

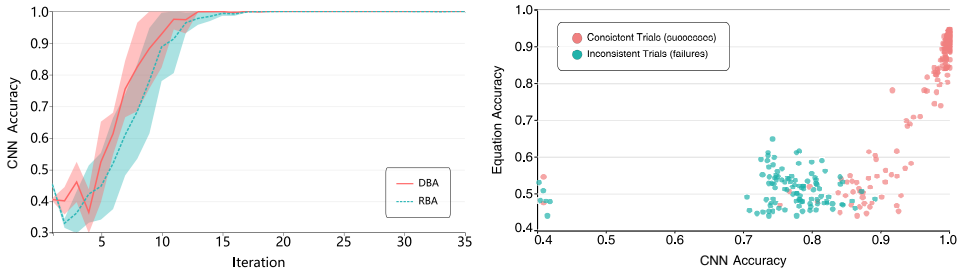
Therefore, the original dataset $D = \{\langle \boldsymbol{x}_i, y_i \rangle\}$ can be transforms into a new dataset $D' = \{\langle \boldsymbol{u}_i, y_i \rangle\}$, from which a decision model is learned to handle the noises introduced by subsampling.

5 Experiments

Dataset We constructed two image sets of symbols to build the equations shown in Fig. 6. The Digital Binary Additive (DBA) equations were created with images from benchmark handwritten character datasets [22, 36], while the Random Symbol Binary Additive (RBA) equations were constructed from randomly selected characters sets of the Omniglot dataset [21] and shared isomorphic structure with the equations in the DBA tasks. In order to evaluate the perceptual generalisation ability of the compared methods, the images for generating the training and test equations are disjoint. Each equation is input as a sequence of raw images of digits and operators. The training and testing data contains equations with lengths from 5 to 26. For each length it contains 300 randomly generated equations, in a total of 6,600 training examples. This task has $4! = 24$ possible mappings from the CNN outputs to the pseudo-label symbols, and $4^3 = 64$ possible operation rule sets (with commutative law), so the search space of logical abduction contains 1536 different possible Δ_C. Furthermore, the abduction for revising pseudo-labels introduces 2^M more candidates. Considering the small amount of training data (especially for the **ABL-short** setting with only 1200 training examples), this task is not trivial.

Compared methods

- **ABL**: The machine learning model of ABL consists of a two-layer CNN and a two-layer multiple-layer perceptron (MLP) followed by a softmax layer; the logical abduction will keep 50 calculation rule sets of bit-wise operations set as relational features; The decision model is a two-layer MLP. Two different settings have been tried: the **ABL-all** that uses all training data and the **ABL-short** that only uses training equations of lengths 5-8.
- **Differentiable Neural Computer** (DNC) [13]: This is a deep neural network associated with memory, and has shown its potential on symbolic computing tasks [13].



(a) CNN training accuracy. (b) Results of logical abduction in RBA tasks.

Figure 8: Training accuracy and results of logical abductions.

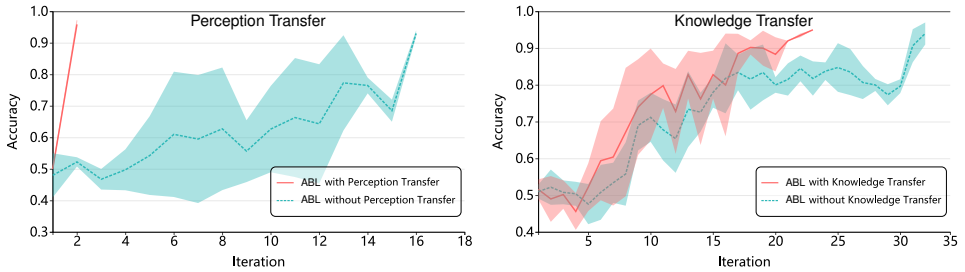


Figure 9: Results of the cross-task transfer experiments.

- **Transformer networks** [39]: This is a deep neural network enhanced with attention, and has been verified to be effective on many natural language processing tasks.
- **Bidirectional Long Short-Term Memory Network** (BiLSTM) [32]: This is the most widely used neural network for learning from sequential data.

To handle image inputs, the BiLSTM, DNC and Transformer networks also use the same structured CNN like the ABLs as their input layers. All the neural networks are tuned with a held-out validation set randomly sampled from the training data. All the experiment are repeated for 10 times and performed on a workstation with a 16 core Intel Xeon CPU @ 2.10GHz, 32 GB memory and a Nvidia Titan Xp GPU.

We also carried out a human experiment. Forty volunteers were asked for classifying images of equations sampled from the same datasets. Before taking the quiz, the domain knowledge about the bit-wise operation was provided as hints, but specific calculation rules are not available — just like the setting for ABL. Instead of using the precisely same setting as the machine learning experiments, we gave the human volunteers a simplified version, which only contains 5 positive and 5 negative equations with lengths ranging from 5-14.

Results Fig. 7 shows that on both tasks, the ABL-based approaches significantly outperform the compared methods, and ABL correctly learned the symbolic rules defining the unknown operations. All the methods performed better on the DBA tasks than RBA, because the symbol images in the DBA task are more easily distinguished. The performance of ABL-all and ABL-short have no significant difference, and the performance of the compared approaches degenerates quickly toward the random-guess line as the length of the testing equations grows, while the ABL-based approaches extrapolates better to the unseen data. An interesting result is that the human performance on the two tasks are very close, and both of them are worse than that of ABL. According to the volunteers, they do not suffer from distinguishing different symbols, but machines are better in checking the consistency of logical theories — in which people are prone to make mistakes. Therefore, machine learning systems should make use of their advantages in logical reasoning.

Inside the learning process of ABL, although no ground-truth labels exist for the images of digits and operators, the CNN training accuracy did increase during the learning process, as shown by Fig. 8a. On the other hand, Fig. 8b shows the relationship between ABL's overall equation classification accuracy, image perception accuracy and results of logical abductions on the RBA tasks, where red dots indicate successful abductions and the blue dots signify failures. This result shows that the training of CNN and the logic-based learning of unknown operation rules indeed mutually benefited each other during the training process.

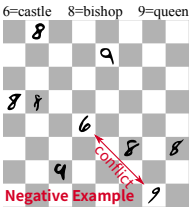
Cross-task Transfer We also carried experiments on transferring the learned CNN and knowledge model (i.e., the relational features Δ_C^t together with the decision MLP) to different tasks.

The first task transfers the CNN learned from the DBA task to logical exclusive-or equations constructed by the same characters. As shown in Fig. 9, although the final performances of ABLs with and without perception transfer are comparable, the convergence of the ABL with perception transfer is much faster. The second task transfers the learned knowledge model from RBA to DBA domains. As depicted in the right side of the same figure, ABL with knowledge transfer converged significantly faster than the compared method. However, comparing the results between knowledge transfer and perception transfer, we can see that machine learning from sub-symbolic data without explicitly providing the labels is considerably more difficult.

6 Discussion

As an important cognitive model in psychology, abduction has already attracted some attention in artificial intelligence [14, 10], while most of existing works combining abduction and induction only consider symbolic domains [37, 7, 29]. There are also some works use abduction to enhance machine learning [4, 24], however, they need to adapt logical background knowledge into functional constraints or use particularly designed operators to support *gradient descent* during learning and reasoning, which relax logical inference into a different continuous optimisation problem.

On the other hand, ABL utilises logical abduction and trial-and-error search to bridge machine learning with original first-order logic, *without* using gradient. As the result, ABL inherits the full power of first-order logical reasoning, e.g., it has the potential of abducing new first-order logical theories that are not in the background knowledge [26]. Consequently, many existing symbolic AI techniques can be directly incorporated without any modification.



(a) Input image.

Method	ABL-ALP	ABL-CLP(FD)
Acc.	0.976 ± 0.008	0.981 ± 0.006
Method	**ABL-CHR**	**CNN**
Acc.	0.979 ± 0.007	0.736 ± 0.016
Method	**Bi-LSTM**	
Acc.	0.513 ± 0.013	

(b) Experimental results.

Figure 10: The extended n-queens experiments, $n \in \{2..10\}$.

In order to verify the flexibility of the ABL framework, a further experiment on the *extended n-queens task* is shown with Fig. 10, whose inputs are images of randomly generated chessboards consist of blanks, queens, castles and bishops represented by randomly sampled MNIST images, and the associated labels are validity of each board. In this task, we implemented logical abduction with Prolog-based ALP and two popular constraint logic programming [16] systems: Constraint Handling Rules [8] and CLP(FD) [3]. Given recursive first-order logical background knowledge about chess moves, the ABL-based approaches achieved better results comparing to CNN and Bi-LSTM.

7 Conclusion

In this paper, we present the *abductive learning*, where machine learning and logical reasoning can be entangled and mutually beneficial. Our initial implementation of the ABL framework shows that it is possible to simultaneously perform *sub-symbolic* machine learning and *full-featured first-order logical reasoning* that allows recursion.

This framework is general and flexible. For example, the perception machine learning model could be a pre-trained model rather than to be learned from scratch; The task for machine learning could be semi-supervised rather than having no label at all; The logical abduction could involve second-order logic clauses to enable abducing recursive clauses and automatically inventing predicates [26]. We hope that the exploration of *abductive learning* will help pave the way to a unified framework accommodating learning and reasoning.

Acknowledgement This research was supported by the National Key R&D Program of China (2018YFB1004300), NSFC (61751306), and the Collaborative Innovation Centre of Novel Software Technology and Industrialisation. The authors would like to thank Yu-Xuan Huang and Le-Wen Cai for their help on experiments, and thank the reviewers for their insightful comments.

References

[1] M. Bošnjak, T. Rocktäschel, J. Naradowsky, and S. Riedel. Programming with a differentiable forth interpreter. In *Proceedings of the 34th International Conference on Machine Learning*, pages 547–556, Sydney, Australia, 2017.

[2] C. P. Bowditch. *The numeration, calendar systems and astronomical knowledge of the Mayas*. Cambridge University Press, Cambridge, UK, 1910.

[3] M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *Proceedings of the International Symposium on Programming Language Implementation and Logic Programming*, pages 191–206, Southampton, UK, 1997. Springer.

[4] W.-Z. Dai and Z.-H. 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. AAAI.

[5] L. De Raedt and A. Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

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

[7] P. A. Flach and A. C. Kakas, editors. *Abduction and Induction*. Springer, Dordrecht, Netherlands, 2000.

[8] T. Frühwirth. Theory and practice of constraint handling rules. *The Journal of Logic Programming*, 37(1):95–138, 1998.

[9] A. S. d. Garcez, K. B. Broda, and D. M. Gabbay. *Neural-symbolic learning systems: foundations and applications*. Springer-Verlag, London, UK, 2012.

[10] A. S. d. Garcez, D. M. Gabbay, O. Ray, and J. Woods. Abductive reasoning in neural-symbolic systems. *Topoi*, 26(1):37–49, Mar 2007.

[11] A. L. Gaunt, M. Brockschmidt, N. Kushman, and D. Tarlow. Differentiable programs with neural libraries. In *Proceedings of the 34th Annual International Conference on Machine Learning*, pages 1213–1222, Sydney, Australia, 2017. ACM.

[12] L. Getoor and B. Taskar, editors. *Introduction to statistical relational learning*. MIT Press, Cambridge, Massachusetts, 2007.

[13] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, and J. Agapiou. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[14] S. Hölldobler, T. Philipp, and C. Wernhard. An abductive model for human reasoning. In *Papers from the 2011 AAAI Spring Symposium, Technical Report SS-11*, Stanford, CA, 2011. AAAI.

[15] S. D. Houston, O. C. Mazariegos, and D. Stuart, editors. *The decipherment of ancient Maya writing*. University of Oklahoma Press, Norman, OK, 2001.

[16] J. Jaffar and M. J. Maher. Constraint logic programming: a survey. *The Journal of Logic Programming*, 19-20:503 – 581, 1994. Special Issue: Ten Years of Logic Programming.

[17] R. Jia and P. Liang. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2011–2021, Copenhagen, Denmark, 2017. ACL.

[18] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic Computation*, 2(6):719–770, 1992.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates Inc., 2012.

[20] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. Mansinghka. Picture: A probabilistic programming language for scene perception. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 4390–4399, Boston, MA, 2015. IEEE.

[21] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[23] L. Magnani. *Abductive Cognition: The Epistemological and Eco-Cognitive Dimensions of Hypothetical Reasoning*. Springer-Verlag, Berlin, German, 1st edition, 2009.

[24] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems 31*, pages 3749–3759. Curran Associates, Inc., 2018.

[25] S. H. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[26] S. H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 2015. Published online: DOI 10.1007/s10994-014-5471-y.

[27] A. Newell and H. A. Simon. The logic theory machine – A complex information processing system. *IRE Transactions on Information Theory*, 2(3):61–79, 1956.

[28] S. C. Peirce. Abduction and induction. In J. Buchler, editor, *Philosophical writings of peirce*. Dover Publications, New York, NY, 1955.

[29] O. Ray. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329 – 340, 2009. Special Issue: Abduction and Induction in Artificial Intelligence.

[30] S. J. Russell. Unifying logic and probability. *Communications of the ACM*, 58(7):88–97, 2015.

[31] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. P. Lillicrap. A simple neural network module for relational reasoning. *CoRR*, abs/1706.01427, 2017.

[32] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[33] H. A. Simon and A. Newell. Human problem solving: The state of the theory in 1970. *American Psychologist*, 26(2):145, 1971.

[34] R. L. Solso, O. H. MacLin, and M. K. MacLin. *Cognitive Psychology*. Pearson/Allyn and Bacon, New York, NY, 8th edition, 2008.

[35] J. B. Tenenbaum, C. Kemp, T. L. Griffiths, and N. D. Goodman. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331(6022):1279–1285, 2011.

[36] M. Thoma. The HASYv2 dataset. *CoRR*, abs/1701.08380, 2017.

[37] C. A. Thompson and R. J. Mooney. Inductive learning for abductive diagnosis. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 664–669, Seattle, WA, 1994. AAAI.

[38] G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1):119 – 165, 1994.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[40] Y. Yu, H. Qian, and Y.-Q. Hu. Derivative-free optimization via classification. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 2286–2292, Phoenix, AZ, 2016. AAAI.

[41] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

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

A Background Knowledge for Handwritten Equations Decipherment

In this section, we present the background knowledge of the Handwritten Equations Decipherment tasks in Prolog language. Following the convention of logic programming, we use words starting with capital letters and underline dash to represent variables, e.g., "Y", "Pseudo_Label" and "_"; and use numbers and words starting with lowercase to represent constant, predicate an function names, e.g., "0", "+" and "digit/1", where the number behind the slash means the arity of the predicate or function. The sentences initiated with "%" are inline comments.

Table 1: Background knowledge about equation structure.

```
% Define a single digit
digit(0).
digit(1).
% Recursively define digits
digits([D]) --> [D],  digit(D) .
digits([D|T]) --> [D], !, digits(T),  digit(D) .
digits(X) :- phrase(digits(X), X).

% Recursively define equations.
% Since the pseudo-labels may contain missing values (variables),.
% we define eq_args as non operator symbols (including variables).
eq_arg([D]) --> [D],  not(D == '+'), not(D == '=') .
eq_arg([D|T]) -->
    [D], !, eq_arg(T),  not(D == '+'), not(D == '=') .
equation(eq(X, Y, Z)) -->
    eq_arg(X), [+], eq_arg(Y), [=], eq_arg(Z).
parse_eq(Pseudo_Labels, Eq) :-
    phrase(equation(Eq), Pseudo_Labels).
```

Tab. 1 illustrates the background knowledge about digits and equation structures. The predicate `digit/1` defines the two numerical pseudo-labels (primitive concepts) "0" and "1", the other two, "+" and "=", are defined in the Definitive Clause Grammar (DCG) rule of `equation/1`.

The predicates in red colour are *recursive* DCG rules, defining the pattern of corresponding concepts. For example, the rules about `digits/1` indicates that any list longer than 1 and constructed by `digit` are "digits".

The `eq_arg/1` predicate simply defines what are the *argument* of `equation/1`. Because the input pseudo-labels may contain missing values (i.e., the "incorrect" pseudo-labels that have been removed by the learned heuristic δ function in Eq. 5), the *argument* could be any list composed by `digit` and Prolog's blank variable "_", for example, "10010" and "_101_1".

The DCG rule for `equation/1` defines that any equation has the structure of `eq(X,Y,Z)`, forming a pseudo-label list "`[X],[+],[Y],[=],[Z]`", in which `X`, `Y` and `Z` are instances of `eq_arg/1`.

The predicates in blue simply parses a list given corresponding concepts defined DCG rules, e.g., `parse_eq/2` takes a list of pseudo-labels as input, and outputs the parsed `equation` structure `eq(X,Y,Z)`.

Parsing a sequence with DCG rules is also a kind of *abduction*, in which the DCG rules are *background knowledge*, the input list is *observation*, and the parsed results are *abduced* explanations. Therefore, there could be multiple parsing results. For example, `parse_eq([A,B,C,D,E,F,G],Eq)` will output `Eq=eq([A,B],[D],[F,G])` or `eq([A],[C,D],[F,G])`, where `C` and `E` are abduced to be "+" and "=" in the first case; `B` and `E` are abduced to be "+" and "=" in the second case.

Tab. 2 shows the background knowledge about bit-wise calculation, which calculates the parsed equation `eq(X,Y,Z)` and abduces the missing pseudo-labels in `eq(X,Y,Z)` as well as the missing operation rules for defining "+". In our implementation, the missing operation rules to be learned, i.e. the Δ_C is defined with "`my_op/3`". For example, a complete rule set defining arithmetic

Table 2: Background knowledge about bit-wise calculation.

```
% Abductive bit-wise calculation with given pseudo-labels,
% this procedure abduces missing pseudo-labels together with
% unknown operation rules.
calc(Rules, Pseudo) :-
    calc([], Rules, Pseudo).
calc(Rules0, Rules1, Pseudo) :-
    parse_eq(Pseudo, eq(X,Y,Z)),
    bitwise_calc(Rules0, Rules1, X, Y, Z).

% Bit-wise calculation that handles carrying
bitwise_calc(Rules, Rules1, X, Y, Z) :-
    reverse(X, X1), reverse(Y, Y1), reverse(Z, Z1),
    bitwise_calc_r(Rules, Rules1, X1, Y1, Z1),
    maplist(digits, [X,Y,Z]).
% Recursively calculate back-to-front
bitwise_calc_r(Rs, Rs, [], Y, Y).
bitwise_calc_r(Rs, Rs, X, [], X).
bitwise_calc_r(Rules, Rules1, [D1|X], [D2|Y], [D3|Z]) :-
    % Abduces Δ_C (my_op/3) during the calculation.
    abduce_op_rule(my_op([D1],[D2],Sum), Rules, Rules2),
    % Handling carry
    ((Sum = [D3], Carry = []); (Sum = [C,D3], Carry = [C])),
    bitwise_calc_r(Rules2, Rules3, X, Carry, X_carried),
    bitwise_calc_r(Rules3, Rules1, X_carried, Y, Z).
```

binary addition that is going to be learned should contain `my_op(0,0,[0])`, `my_op(0,1,[1])`, `my_op(1,0,[1])` and `my_op(1,1,[1,0])`. However, in the experiments the ABL sometimes output `my_op(1,1,[1])`, `my_op(0,1,[0])`, `my_op(1,0,[0])` and `my_op(0,0,[0,1])`, which flips the semantics of `0` and `1`.

The `calc/2` takes a list of pseudo-labels as input, and outputs the possible Δ_C `Rules` and the missing pseudo-labels that have been removed by the δ function. `calc/3` function is a more flexible version of `calc/2`, which is able to take some already abduced operation rules `Rules0` in to consideration.

The `bitwise_calc/5` defines the abductive bit-wise calculation process. Given an initialised operation rule set `Rules0` (usually the empty ruleset "`[]`" according to `calc/2`), it abduces the consistent operation rule set `Rules1` with revised pseudo-labels `X`, `Y` and `Z`. Firstly, it calls the `reverse/2` predicate to reverse the equation arguments, then calls the reverse bit-wise calculation predicate `bitwise_calc_r/5` to complete the abduction, and finally check if the abduced pseudo-label forms legitimate `digits`.

The `bitwise_calc_r/5` predicate calculates `X+Y` to `Z` bit by bit. It terminates when `X` or `Y` runs out of digits and fails if there exists no consistent operation rules with pseudo-labels. During the calculation, it calls the predicate `abduce_op_rule/3` from the Abductive Logic Program (shown in Tab. 3) to abduce consistent operation rules. It also allows 1-digit carry in its calculation.

Tab. 3 defines the Abductive Logic Program (ALP) for the logical abduction in our handwritten equation decipherment tasks.

`abduce/2` is ABL's main predicate for making abduction. Given a set of examples that has been interpreted by the perception machine learning model (and with the learned δ function marking out the incorrect pseudo-labels), `abduce/2` will try to abduce a Δ_C and the revised pseudo-labels consistent with all the background knowledge and the labels about the target concept of the examples.

The `abduce/3` predicate processes examples sequentially. By abductively calculating the examples one-by-one, it not along abduces the missing pseudo-labels in each example, but also continuously put consistent bit-wise operation rule in Δ_C by calling the `calc/3` predicate defined in Tab. 2.

Table 3: The Abductive Logic Program for handwritten equation decipherment.

```
% Main predicate for peforming abduction
% ‘‘Examples’’ are the pseudo-labels of a set of examples,
% ‘‘Delta_C’’ is the abduced Δ_C.
abduce(Examples, Delta_C) :-
    abduce(Examples, [], Delta_C).
abduce([], Delta_C, Delta_C).
abduce([E|Examples], Delta_C0, Delta_C1) :-
    calc(Delta_C0, Delta_C2, E),
    abduce(Exs, Delta_C2, Delta_C1).

% Getting an existed (already abduced) operation rule from history.
abduce_op_rule(R, Rules, Rules) :-
    member(R, Rules).
% Abduce a new rule.
abduce_op_rule(R, Rules, [R|Rules]) :-
    op_rule(R),
    % integrity constraints.
    valid_rules(Rules, R).

% Integrity Constraints on operation rule set, forbidding
% redundant rules and inconsistent rules.
valid_rules([], _).
valid_rules([my_op([X1],[Y1],_)|Rs], my_op([X],[Y],Z)) :-
    not([X,Y] = [X1,Y1]),
    not([X,Y] = [Y1,X1]),
    valid_rules(Rs, my_op([X],[Y],Z)).
valid_rules([my_op([Y],[X],Z)|Rs], my_op([X],[Y],Z)) :-
    not(X = Y),
    valid_rules(Rs, my_op([X],[Y],Z)).

% Abducing single operation rule.
op_rule(my_op([X],[Y],[Z])) :- digit(X), digit(Y), digit(Z).
op_rule(my_op([X],[Y],[Z1,Z2])) :- digit(X), digit(Y), digits([Z1,Z2]).
```

The `abduce_op_rule/3` called by `bitwise_calc_r/5` can abduce one operation rule in each call. Before doing the abduction, it first returns an already abduced operation rule `R` in `Rules` and let `bitwise_calc_r/5` to determine if it is consistent with current calculation. If a history `R` already meets the requirement then it does nothing; otherwise it will try to abduce a `my_op/3` rule defined by `op_rule/1` and return it to the calculation process of `bitwise_calc_r/5`.

During the abduction process, `abduce_op_rule/3` will call an integrity constraint `valid_rules/2` to test if the newly abduced `R` is consistent with previously abduced rule set `Rules`. Basically, the integrity constraint says that:

1. No redundant operation rules, i.e., there shouldn’t be two separate rules defining the same operation `X+Y`;
2. No conflict operation rules according to the *commutative law*, i.e., `X+Y=Y+X`.

B Errors cases of ABL in the experiments

The failures of ABL-based systems are mostly caused by perception errors. Fig. 11 shows one of the failure examples in the RBA task:

The ground-truth symbols in the equation is “110010+11100110=100011000”, but the perceived result by ABL is “110010+11100110110001=000”, causing a classification failure. After examined

Figure 11: An example of the wrongly predicted equations during test.

the experimental results, we found that almost all of the learned operation rule sets Δ_C^t (relational features) are correct, and the equation classification errors are only caused by the incorrectly perceived pseudo-labels. In fact, Fig. 8b has shown that the performance of ABL relies much on perception accuracy. More interestingly, according to the human volunteers, the failures made by them are majorly caused by reasoning errors, i.e. the difficulties in finding consistent operation rules, which is opposite to ABL.