ORIGINAL ARTICLE

Logic could be learned from images

Qian Guo[1,3] · Yuhua Qian[1,2,3] · Xinyan Liang[1,3] · Yanhong She[4] · Deyu Li[2,3] · Jiye Liang[2,3]

Abstract

Logic reasoning is a significant ability of human intelligence and also an important task in artificial intelligence. The existing logic reasoning methods, quite often, need to design some reasoning patterns beforehand. This has led to an interesting question: can logic reasoning patterns be directly learned from given data? The problem is termed as a data concept logic. In this study, a learning logic task from images, called a LiLi task, first is proposed. This task is to learn and reason the logic relation from images, without presetting any reasoning patterns. As a preliminary exploration, we design six LiLi data sets (Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication), in which each image is embedded with a n-digit number. It is worth noting that a learning model beforehand does not know the meaning of the n-digit numbers embedded in images and the relation between the input images and the output image. In order to tackle the task, in this work we use many typical neural network models and produce fruitful results. However, these models have the poor performances on the difficult logic task. For furthermore addressing this task, a novel network framework called a divide and conquer model by adding some label information is designed, achieving a high testing accuracy.

Keywords Logic reasoning · Data concept logic · LiLi task · Reasoning patterns

✉ Yuhua Qian
 jinchengqyh@126.com

 Qian Guo
 czguoqian@163.com

 Xinyan Liang
 liangxinyan48@163.com

 Yanhong She
 yanhongshe@gmail.com; yanhongshe@xsyu.edu.cn

 Deyu Li
 lidy@sxu.edu.cn

 Jiye Liang
 ljy@sxu.edu.cn

1   Institute of Big Data Science and Industry, Shanxi
    University, Taiyuan 030006, Shanxi, China

2   Key Laboratory of Computational Intelligence and Chinese
    Information Processing of Ministry of Education, Shanxi
    University, Taiyuan 030006, Shanxi, China

3   School of Computer and Information Technology, Shanxi
    University, Taiyuan 030006, Shanxi, China

4   College of Science, Xi'an Shiyou University, Xi'an 710065,
    Shanxi, China

1 Introduction

Human intelligence integrates cognitive functions such as perception, learning, memory, problem solving and logic reasoning [5]. Among them, logic reasoning is a significant ability of human intelligence. Applying the reasoning, humans obtain some rules hidden in complex phenomenon, and even forecast the unknown events. One of the goals of artificial intelligence is to mimic human cognitive functions to the utmost. As a part of cognitive functions, logic reasoning is also an important task in artificial intelligence [19].

Many logic reasoning methods such as fuzzy reasoning [27, 31, 46, 51], FCA [7, 37, 42, 47], probabilistic reasoning [23, 28, 29, 39], evidential reasoning [4, 30], Bayesian reasoning [44, 49] and rough reasoning [22, 26, 34, 38], have been proposed. However, quite often, these methods need to design some reasoning patterns beforehand. For example, in the FCA, one first obtains a formal context applying the domain expert knowledge, then computes the concept lattice from the formal context, and finally achieves knowledge reasoning using the disjunction and conjunction operations. This process not only costs a large amount of time, but also heavily depends on the domain expert experience. But, without mastering special

domain knowledge beforehand, human still can directly reason from given data. For example, without mastering knowledge of 3D reconstruction beforehand, people can reconstruct 3D model of an unseen 2D image in his mind through observing and reasoning many 2D images and corresponding 3D scenes in real world. This has led to an interesting research topic: can machine directly learn logic reasoning patterns from given data? And these logical patterns are termed as the data concept logic (DCL).

As a preliminary exploration, in this study, we design a task of the DCL which is called learning logic task from images, just a LiLi task shown in Fig. 1(7). Unlike the logical operation defined by human (LOH) using some domain expert knowledge, a LiLi task is to learn and reason the relation between two input images and one output image without any reasoning patterns beforehand, i.e. LiLi does not know any reasoning patterns about R. In summary, there are some differences below between a LiLi task and a LOH.

- For LiLi, one does not know any reasoning patterns about R except for giving a data set, while for LOH whose focus is that how to define a reasonable logical operation, one always possesses lots of domain knowledge about R.
- LPN induced by a LiLi models an abstract or low level logical relation in term of the pixel values. However, the

existing logical operation models a semantic or high level logical relation in term of the numbers or symbols.
- LPN induced by a LiLi is a data-driven method to model the logical relation, while LOH is an expert-driven method.

Learning logic task from images (LiLi task) is also a very important computer vision task. Unfortunately, to the best of our knowledge, there are only a bit of work on the LiLi task shown in Fig. 1(7). [10] mined the logical patterns from Fashion-Logic data sets without presetting any reasoning patterns. Zhou et al. [6] proposed a Neural Logical Machine (NLM) which can learn new logic based on some background knowledge. In contrast, a variety of models based on deep convolutional neural networks (CNNs) have achieved the state-of-the-art performances, even super-human on some tasks for the common computer vision tasks such as object recognition [11, 17, 25], object detection [12, 36], semantic segmentation [3, 40], image captioning [18, 45], visual question answering (VQA) [48, 50], image generator [8, 35] (see Fig. 1). It is well known that the logic reasoning is one of the abilities that the general/strong artificial intelligence has to possess. In the existing computer vision tasks, image captioning and visual question answering seem to need some reasoning abilities, especially VQA (indeed
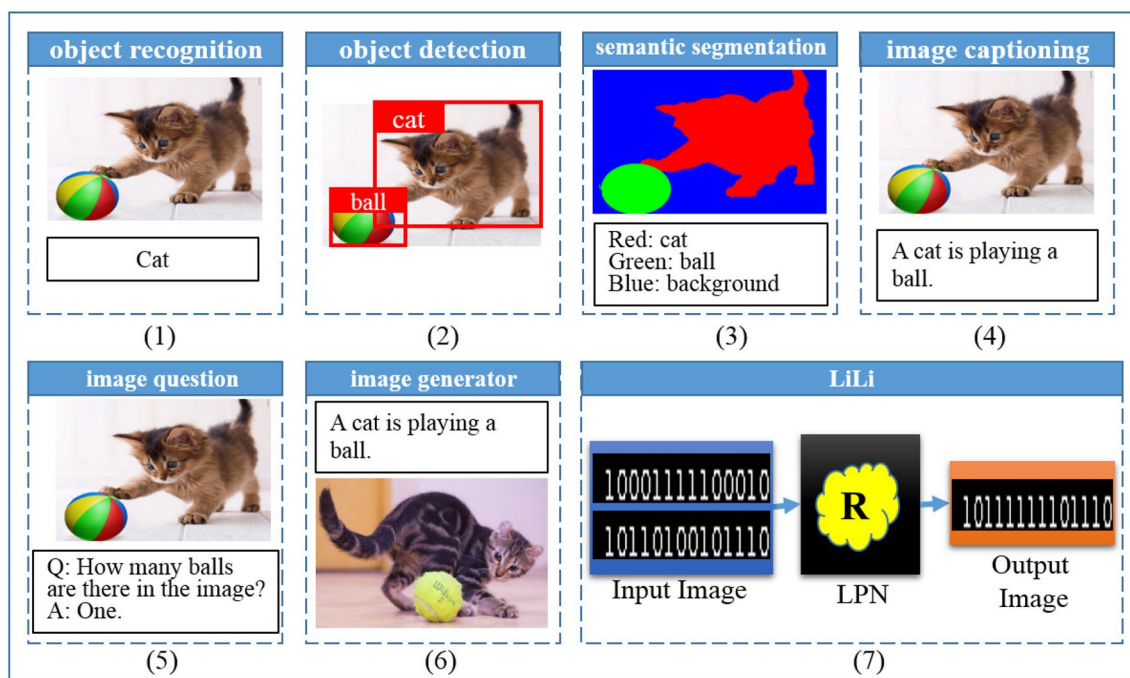
Fig. 1 The differences among these popular computer vision tasks. (1) Object recognition (sometimes object classification) is to classify individual objects. (2) Object detection is to classify individual objects and localize each using a bounding box. (3) Semantic segmentation is to classify each pixel into a fixed set of categories without differentiating object instances. (4) Image captioning is to describe the content of an image by using reasonably formed natural sentences. (5) Visual question answering (VQA) is to automatically answer natural language questions according to related the image content. (6) Image generator is to generate images according to images or text description. (7) Data concept logic is to learn to obtain logic concepts from a given data set

VQA performs need more knowledge: image itself, common sense, domain knowledge, and so on). In fact, because of some shortcomings of existing benchmark data sets (described in Sect. 3.1), the systems can correctly answer the questions without reasoning [16, 19, 53]. Hence, it is desired to provide a new task, such as the LiLi task, to test the reasoning ability of models.

Our contributions are as follows:

1. The data concept logic (DCL) is proposed to directly learn the concept logical patterns from the given data.
2. We propose a LiLi task where the abstract or low logical relation between two input images and one output image needs to be learned and reasoned without any reasoning patterns beforehand.
3. We provide an inference form of the LiLi task that is the consistent with classical propositional calculus form.
4. Six LiLi data sets with three difficulty levels: Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication, are provided.
5. Unlike a semantic or high level logical relation defined by human, an abstract or low level logical relation is expressed by a novel data-driven method called as LPN.
6. The performances of these typical neural networks: CNN-LSTM, MLP, CNN-MLP, autoencoder and ResNets, are tested on six LiLi data sets.
7. The divide and conquer model (DCM) is proposed using a decomposing strategy to solve the difficult task multiplication, achieving a better performance than the typical neural networks used in this paper.

The remainder of this paper is organized as follows: Sect. 2 proposes the DCL. Section 3 proposes six LiLi data sets, the LiLi task and its inference form. Section 4 presents the performance evaluation of the typical neural networks on six LiLi data sets. In Sect. 5, the DCM is devised to solve the difficult logic task multiplication. Finally, we draw conclusions in Sect. 6.

2 DCL

In this section, we first detail the DCL proposed in this paper, and then provide an inference form of DCL.

2.1 DCL

Data concept logic (DCL) is a data-driven tool for learning to obtain logic concepts from a given data set directly. Applying the learned concepts, it can output the logical relations among the input data. It is noted that DCL merely uses pure original data cues, and can not know other information such as the meaning of symbols/numbers in data in advance. The DCL can be formalized as follows.

Definition 1 A data concept logic is termed as a triple $\mathcal{R} = (I, R, O)$, where $I = \{x_i \mid x_i = (x_i^1, x_i^2, \ldots, x_i^{m_I}), i = 1, 2, \ldots, N\}$ is an input sequence with the length m_I, $O = \{y_i \mid y_i = (y_i^1, y_i^2, \ldots, y_i^{m_O}), i = 1, 2, \ldots, N\}$ is an output sequence with the length m_O, $R : I \to O$ is a reasoning pattern (relation mapping) from the input I to the output O.

The aim of DCL is to learn the R from the input I to the output O. In this paper, we propose a deep learning network framework: Logical Pattern Network (*LPN*) parameterized by W to learn the R. This model can be learned by solving the following optimization problem.

$$
\begin{aligned}
W^* &= \arg\min_W \mathcal{L}(LPN_W(I), O) \\
&= \arg\min_W \frac{1}{N} \sum_{i=1}^N \mathcal{L}(LPN_W(x_i), y_i),
\end{aligned}
\tag{1}
$$

where $\mathcal{L}$ is a loss function, and N is the number of the training samples.

The universal approximation theorem tells us that neural networks are able to approximate any measurable function with any precision [14]. Theoretically, the logical pattern R can be represented by one neural network. In the DCL, R is hidden in the LPN, and mining R from data can be regarded as the iterative optimization process of parameter W of LPN. At each iteration, the value of W changes in the direction that the loss $\mathcal{L}$ becomes smaller. When the loss is small enough, the iteration stops and R is obtained.

The workflow of a DCL task is illustrated in Fig. 2, where I is the set of input data, O is the set of ground-truth output data, $\hat{O}$ indicates the set of logical relation patterns reasoned by $f(LPN_W(x_i^1, x_i^2, \ldots, x_i^{m_I}))$, O/I is the ground-truth logical relation set for a given input set I, $\hat{O}/I$ is the prediction logical relation set for a given input set I using *LPN*, Loss is used to evaluate the difference between O/I and $\hat{O}/I$. *LPN* indicates the logical pattern network.
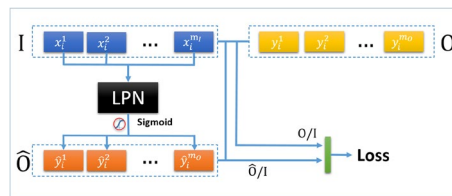

Fig. 2 The workflow of a DCL task

2.2 Inference form of DCL

Human, in our daily life, often makes inferences using some known antecedents. And this process can be formalized as the following form [27].

$$
\begin{aligned}
&Antecedent\ 1:\ A_1\ \longrightarrow B_1\\
&Antecedent\ 2:\ A_2\ \longrightarrow B_2\\
&\qquad \vdots \qquad\quad \vdots\\
&Antecedent\ n:\ A_n\ \longrightarrow B_n\\
&Antecedent\ *:\ A_*\\
\hline
&Consequence:\qquad\qquad B_*,
\end{aligned} \tag{2}
$$

Formula (2) exactly is also the mathematical model of the classical propositional calculus [27] where the consequence of the antecedent $*$ is inferred using the known n antecedents. There exist many methods addressing the task. For example, Zadeh [52] provided an inference rule called 'compositional rule of inference' (CRI) to make such an inference whose antecedents and consequences contain fuzzy concepts. Specially, an implication $A \to B$ first is translated into a fuzzy relation $R_z(A, B)$ from A to B. And then, B_* can be inferred by the composition of R_z and A_* by the following formula.

$$
B_* = R_z(A, B) \circ A_*, \tag{3}
$$

where $R_z : [0, 1]^2 \to [0, 1]$ defined beforehand by the human experts is a duality function. $\circ$ denotes a composition operator.

Inspired by fuzzy reasoning [27], a DCL task can be written as the following inference form based on the IF THEN rule.

$$
\begin{aligned}
&Antecedent\ 1: && \textit{If the input is } x_1 \textit{ then the output is } y_1\\
&Antecedent\ 2: && \textit{If the input is } x_2 \textit{ then the output is } y_2\\
&\qquad \vdots && \qquad\quad \vdots\\
&Antecedent\ n: && \textit{If the input is } x_n \textit{ then the output is } y_n\\
&Antecedent\ n+1: && \textit{If the input is } x_{n+1}\\
&Antecedent\ n+2: && \textit{If the input is } x_{n+2}\\
&\qquad \vdots && \qquad\quad \vdots\\
&Antecedent\ n+m: && \textit{If the input is } x_{n+m}\\
\hline
&Consequence\ n+1: && \textit{The output is } y_{n+1}\\
&Consequence\ n+2: && \textit{The output is } y_{n+2}\\
&\qquad \vdots && \qquad\quad \vdots\\
&Consequence\ n+m: && \textit{The output is } y_{n+m},
\end{aligned} \tag{4}
$$

where x_i is the input of the LPN, y_i is the output of the LPN.

It should be noted that x_i and y_i can be many kinds of objects in LPN. For example, antecedents and consequences contain fuzzy concepts as shown in Formula (2).

In this paper, x_i and y_i are images. Specifically, x_i is an input sequence with the length m_I, y_i is an output sequence with the length m_O. Based on this, Formula (4) can be written as the following form.

$$
\begin{aligned}
&Antecedent\ 1: && \textit{If the input squence is } (x_1^1, x_1^2, \ldots, x_1^{m_I})\\
&&& \textit{then the output squence is } (y_1^1, y_1^2, \ldots, y_1^{m_O})\\
&Antecedent\ 2: && \textit{If the input squence is } (x_2^1, x_2^2, \ldots, x_2^{m_I})\\
&&& \textit{then the output squence is } (y_2^1, y_2^2, \ldots, y_2^{m_O})\\
&\qquad \vdots && \qquad\qquad \vdots\\
&Antecedent\ n: && \textit{If the input squence is } (x_n^1, x_n^2, \ldots, x_n^{m_I})\\
&&& \textit{then the output squence is } (y_n^1, y_n^2, \ldots, y_n^{m_O})\\
&Antecedent\ n+1: && \textit{If the input squence is } (x_{n+1}^1, x_{n+1}^2, \ldots, x_{n+1}^{m_I})\\
&Antecedent\ n+2: && \textit{If the input squence is } (x_{n+2}^1, x_{n+2}^2, \ldots, x_{n+2}^{m_I})\\
&\qquad \vdots && \qquad\qquad \vdots\\
&Antecedent\ n+m: && \textit{If the input squence is } (x_{n+m}^1, x_{n+m}^2, \ldots, x_{n+m}^{m_I})\\
\hline
&Consequence\ n+1: && \textit{The output squence is } (y_{n+1}^1, y_{n+1}^2, \ldots, y_{n+1}^{m_O})\\
&Consequence\ n+2: && \textit{The output squence is } (y_{n+2}^1, y_{n+2}^2, \ldots, y_{n+2}^{m_O})\\
&\qquad \vdots && \qquad\qquad \vdots\\
&Consequence\ n+m: && \textit{The output squence is } (y_{n+m}^1, y_{n+m}^2, \ldots, y_{n+m}^{m_O}),
\end{aligned} \tag{5}
$$

where $(x_i^1, x_i^2, \ldots, x_i^{m_I})$ is the input data fed into the LPN, $(y_i^1, y_i^2, \ldots, y_i^{m_O})$ is the output data expressing the relation of the input data.

In Formula (5), the n antecedents from 1 to n constituting the training set are used to train the LPN inference model. And the m antecedents from $n + 1$ to $n + m$ constituting the testing set are used to test the inference ability of LPN. Based on this, Formula (5) can be further simplified as the following form.

$$
\begin{aligned}
&Training\ antecedent: && (x_1^1, x_1^2, \ldots, x_1^{m_I}) && \longrightarrow (y_1^1, y_1^2, \ldots, y_1^{m_O})\\
&&& (x_2^1, x_2^2, \ldots, x_2^{m_I}) && \longrightarrow (y_2^1, y_2^2, \ldots, y_2^{m_O})\\
&&& \qquad \vdots && \qquad \vdots\\
&&& (x_n^1, x_n^2, \ldots, x_n^{m_I}) && \longrightarrow (y_n^1, y_n^2, \ldots, y_n^{m_O})\\
&Testing\ antecedent: && (x_{n+1}^1, x_{n+1}^2, \ldots, x_{n+1}^{m_I})\\
&&& (x_{n+2}^1, x_{n+2}^2, \ldots, x_{n+2}^{m_I})\\
&&& \qquad \vdots\\
&&& (x_{n+m}^1, x_{n+m}^2, \ldots, x_{n+m}^{m_I})\\
\hline
&Consequence: &&&& (y_{n+1}^1, y_{n+1}^2, \ldots, y_{n+1}^{m_O})\\
&&&&& (y_{n+2}^1, y_{n+2}^2, \ldots, y_{n+2}^{m_O})\\
&&&&& \qquad \vdots\\
&&&&& (y_{n+m}^1, y_{n+m}^2, \ldots, y_{n+m}^{m_O}),
\end{aligned} \tag{6}
$$

Formula (6) can be further simplified as the following form by $I_{train} = \{(x_1^1, x_1^2, \ldots, x_1^{m_I}), (x_2^1, x_2^2, \ldots, x_2^{m_I}), \ldots, (x_n^1, x_n^2, \ldots, x_n^{m_I})\}$, $O_{train} = \{(y_1^1, y_1^2, \ldots, y_1^{m_O}), (y_2^1, y_2^2, \ldots, y_2^{m_O}), \ldots, (y_n^1, y_n^2, \ldots, y_n^{m_O})\}$, $I_{test} = \{(x_{n+1}^1, x_{n+1}^2, \ldots, x_{n+1}^{m_I}), (x_{n+2}^1, x_{n+2}^2, \ldots, x_{n+2}^{m_I}), \ldots, (x_{n+m}^1, x_{n+m}^2, \ldots, x_{n+m}^{m_I})\}$, and $O_{test} = \{(y_{n+1}^1, y_{n+1}^2, \ldots, y_{n+1}^{m_O}), (y_{n+2}^1, y_{n+2}^2, \ldots, y_{n+2}^{m_O}), \ldots, (y_{n+m}^1, y_{n+m}^2, \ldots, y_{n+m}^{m_O})\}$

$$
\begin{aligned}
&Training\ antecedent\ set:\ I_{train}\ \longrightarrow O_{train}\\
&Testing\ antecedent\ set:\ I_{test}\\
\hline
&Consequence\ set:\qquad\qquad O_{test},
\end{aligned} \tag{7}
$$

In fact, Formula (7) contains three implications, i.e. $(I_{train} \to O_{train}) \to (I_{test} \to O_{test})$. One can obtain the consequence O_{test} of the antecedent I_{test} by translating three implications to the following form.

$$O_{test} = R(I_{train}, O_{train}) \circ I_{test}, \tag{8}$$

where $R(I_{train}, O_{train})$ learned using a data-driven method is a high-dimension function.

From the above analysis, one can find that the DCL has the consistent inference form with the classical propositional calculus. The comparison of the DCL and the LOH is illustrated in Fig. 3. From Fig. 3, one can see that one fundamental task of DCL or LOH is to obtain the relation R. For this task, they have a very obvious difference: for LOH, R needs to be defined beforehand by the experts, while for DCL, R is learned from a given data set.

Based on the above analysis, it is desired to design a human-free and data-driven method directly learn the reasoning pattern from given data. In this study, we explore this problem by proposing the LiLi task. What follows, the LiLi task will be detailed and formalized.

3 A LiLi task

In this section, we first construct six LiLi data sets, then detail the LiLi task proposed in this paper, and finally provide its inference form consistent with the classical propositional calculus form.

3.1 LiLi data sets

The existing logic reasoning data sets such as CLEVR [19] and VQA [1] have made outstanding contributions to testing the logic reasoning ability of machines, but they have also some shortcomings. (1) Because of biases of the data sets, some questions can be answered through directly perceiving images rather than reasoning [16, 19, 53]. For example, the question is what color is the object in the given image, and the answer can be obtained directly from the image through perception. (2) The existing logic reasoning data sets may seem complex, but the typical neural networks and their results suggest that the logics that are embedded in these data sets are relatively simple for machines. More difficult logic data sets should be designed. (3) Some questions from the existing logic reasoning data sets have multiple answers, so it is not easy to judge whether the answers of these questions are correct or not. These shortcomings make it difficult to assess the reasoning abilities of machines using these data sets.

Therefore, we construct the LiLi data sets to overcome these shortcomings. In this paper, these logical relations: Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication are selected to construct the LiLi data sets. (1) Questions are able to be answered only if a model has both perception and reasoning abilities. (2) The typical neural networks are almost powerless for the logic of multiplication (detailed in Sect. 4). It indicates that the LiLi task is really worth studying. (3) The construction process of the LiLi data set is controlled by us and only one correct answer can be obtained from each sample. Hence, it is easy to evaluate the correctness of the answer.

We construct the LiLi data sets to verity the performance of the proposed LPN model. It is worth noting that the LPN model does not know the logical relations hidden in images beforehand. The bitwise operations are binary numbers and arithmetic operations are decimals. For Bitwise And, Bitwise Or and Bitwise Xor data sets, the size of the images is set to 15×120, so the number embedded in one image is at most a 14-digit number. For Addition, Subtraction and Multiplication data sets, the size of the images is set to 15×60, hence the number embedded in

Fig. 3 The comparative analysis between the DCL and the LOH
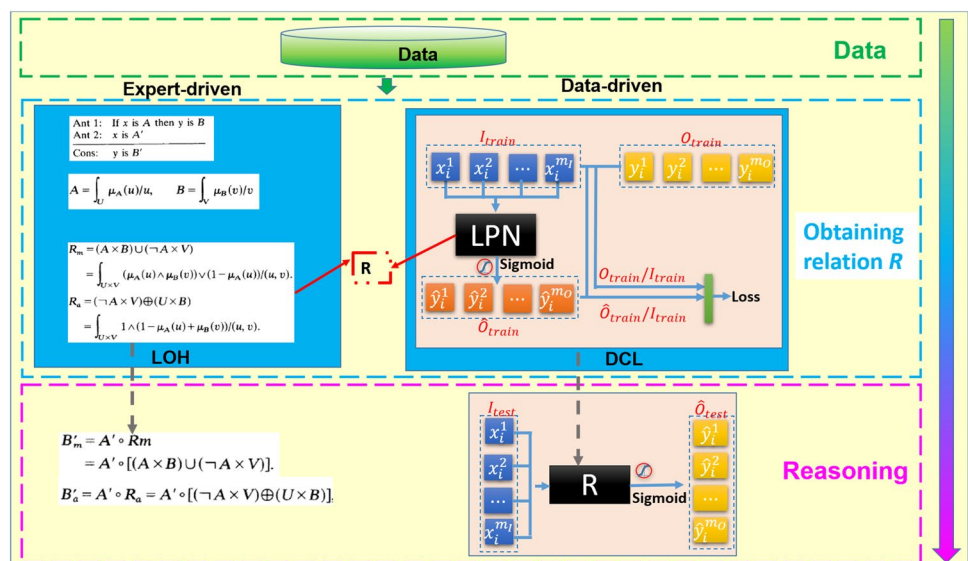
one image is at most a 7-digit number. This step ensures that the proportion of numbers used for training is a very small fraction of all possible combinations. Each of these samples consists of two input images each containing an integer number. The pair of two input images marked x_i^1 and x_i^2 are then generated from a pre-specified range as detailed below. The output image marked y_i is generated according to the result of the operation on the two input images. The numbers embedded in images x_i^1, x_i^2 and y_i are A, B and E.

The details about these data sets are here:

- Bitwise And: For per sample, both A and B have 14 binary digits. E is the bitwise *and* of A and B. For example, A *and* B are "00111101110111" and "10010101110000", respectively. So, E is "00010101110000". The sample is shown in Fig. 4a.
- Bitwise Or: For per sample, both A and B have 14 binary digits. E is the bitwise *or* of A and B. For example, A and B are "10001111100010" and "10110100101110", respectively. So, E is "10111111101110". The sample is shown in Fig. 4b.
- Bitwise Xor: For per sample, both A and B have 14 binary digits. E is the bitwise *xor* of A and B. For example, A and B are "00110101010110" and "00111101110000", respectively. So, E is "00001000100110". The sample is shown in Fig. 4c.
- Addition: For per sample, the range of A and B are 0–4,999,999. E is the sum of A and B. For example, A and B are "646,724" and "4,087,801", respectively. So, E is "4,734,525". The sample is shown in Fig. 4d.
- Subtraction: For per sample, the range of A and B are 0–9,999,999. E is the difference between A and B. In order to ensure a positive result, A is chosen to be larger or equal to B. For example, A and B are "6,740,693" and "3,502,317", respectively. So, E is "3,238,376". The sample is shown in Fig. 4e.
- Multiplication: For per sample, the range of A and B are 0–3160. E is the product of A and B. For example, A and B are "1257" and "1377", respectively. So, E is "1,730,889". The sample is shown in Fig. 4f.

According to the difficulty of the logical relations embedded in data sets, these data sets are divided into 3 levels: one-star (★, easy), two-star (★★, intermediate), and three-star (★★★, difficult).

Bitwise And, Bitwise Or and Bitwise Xor data sets (★): (1) the value of each digit of E is only determined by the values at the same position in A and B, e.g., in Fig. 4a, the value at 2*nd* (the rightmost position is 1*st*) position in E is only determined by the values at 2*nd* position in A and B, so the value at 2*nd* position in E is " 0" (1 & 0 = 0); (2) the possible value of each digit of E is 0 or 1.

Addition and Subtraction data sets (★★): (1) the value of each digit of E is determined by the carry or borrow and the values at the same position in A and B, e.g., in Fig. 4d, the value at 2*nd* position in E is determined by the carry of the sum of values at 1*st* position in A and B and the values at 2*nd* position in A and B; (2) the possible value of carry or borrow part is 0 or 1, so the possible value of each digit (except the rightmost position) of E has two possibilities in 0–9, we choose one of the two possibilities as the final result based on the carry or borrow case. E.g., in Fig. 4d, the carry of the sum of values at 1*st* position in A and B is "0", the sum of values at 2*nd* position in A and B is "2" (2 + 0=2), so the value at 2*nd* position in E is "2" (0 + 2=2).

Multiplication data set (★★★): (1) the value at a given position in E is determined by the values at the given positions in A and B and all positions in A and B before that given position. E.g., in Fig. 4f, the value at 2*nd* position in E is determined by the values at 1*st* and 2*nd* positions in A and the values at 1*st* and 2*nd* positions in B. (2) The number of the possible value of each digit (except the rightmost position) of the E on Multiplication data set is more than that on other LiLi data sets.

3.2 LiLi task

In this paper, we focus on the scene where a model directly learns and reasons the relation between two input images and one output image, without any reasoning patterns beforehand. In this task, we first generate three images, two for the input and one for the output. The output image expresses the

Fig. 4 The samples of six LiLi data sets

(a) Bitwise And

(b) Bitwise Or

(c) Bitwise Xor

(d) Addition

(e) Subtraction

(f) Multiplication
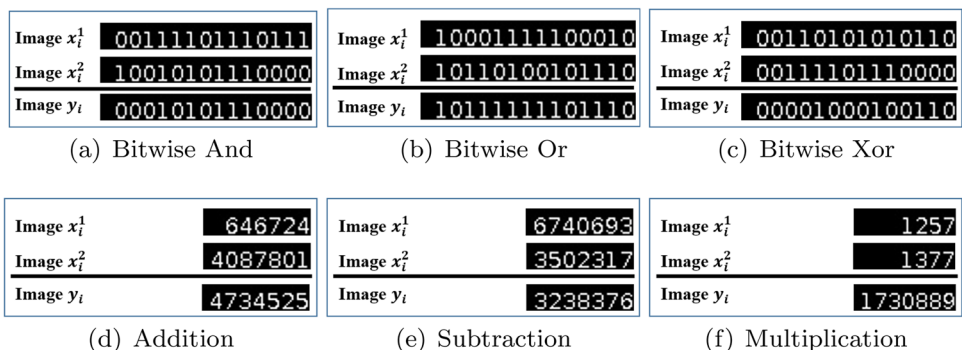
relation between two input images. In addition, the n-digit number embedded in the images are not explicitly introduced, which means that the meaning of contents embedded in images and the relation between two input images and one output image are not known at all. One example is used to illustrate the LiLi task. If the n-digit numbers embedded in two input images are "234" and "432", the output image are "666", the logical relation between two input images and the output image is addition. It can be formalized as follows.

Given a data concept logic system as a set of triple $\mathcal{R} = (I, R, O)$, where $I = \{x_i \mid x_i = (x_i^1, x_i^2), i = 1, 2, \ldots, N\}$ is an input sequence, $O = \{y_i\}_{i=1}^N$ is the output sequence, where x_i^1, x_i^2 and y_i are three images with K pixels shown in Fig. 4. R denotes the logical relation between the pair of images $x_i \in I$ and $y_i \in O$.

At the semantic or high level, R is called as *Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication* denoted as $\&, |, \wedge, +, -$ or $\times$, and they are easily understood by human beings. However, at the abstract or low level, R may be a high-dimensional mapping that is extremely difficult to define the mapping by human, in this paper, $R : [-1, 1]^{2K} \to \{0, 1\}^K$. Hence, it is desired to design a novel method to express an abstract or low level logical relation.

In this task, given a data set $D = \{(x_i, y_i)\}_{i=1}^N$, where y_i denotes the logical relation between the pair of images x_i^1 and x_i^2. When drawing these images, we use the pixel value 0 for black, the pixel value 1 for white. For the input images, we scale every pixel value into -1 to 1 by subtracting the mean, so $x_i^1, x_i^2 \in [-1, 1]^K$. For the output image $y_i \in \{0, 1\}^K$. This task can be viewed as finding a mapping from the input space $I = \{x_i\}_{i=1}^N$ to the output space $O = \{y_i\}_{i=1}^N$ by a supervised learning strategy. In this study, this task can be transformed into a regression problem with the Mean Square Error (MSE) loss function, i.e. $\mathcal{L}$ is *MSE*. It can be by solving the following optimization problem.

$$
\begin{aligned}
W^* &= \arg\min_W MSE(f(LPN_W(I)), O) \\
&= \arg\min_W \frac{1}{N} \sum_{i=1}^N MSE(f(LPN_W(x_i^1, x_i^2)), y_i) \\
&= \arg\min_W \frac{1}{N} \sum_{i=1}^N \sqrt{\sum_{k=1}^K (f(LPN_W(x_i^1, x_i^2))_k - y_{ik})^2},
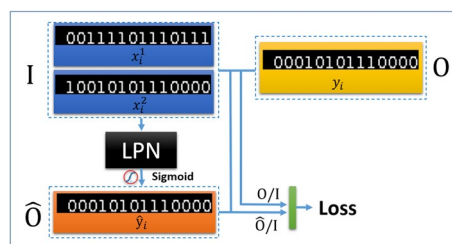\end{aligned} \tag{9}
$$

Fig. 5 The workflow of a LiLi task

where f is a sigmoid function to transform $LPN_W(x_i^1, x_i^2)$ to $[0,1]$, i.e.

$f(LPN_W(x_i^1, x_i^2)) \in [0, 1]^K$, and LPN is parameterized by W. Formula (9) is differentiable with respect to the parameter W, and can be efficiently solved by using the gradient descent method.

Based on above analysis and discussion, we illustrate the workflow of the LiLi task shown in Fig. 5, where I is the set of input image data, O is the set of ground-truth output image data, $\hat{O}$ indicates the set of logical relation patterns reasoned by $f(LPN_W(x_i^1, x_i^2))$, O/I is the ground-truth logical relation set for a given input image set I, $\hat{O}/I$ is the prediction logical relation set for a given input image set I using LPN, Loss is used to evaluate the difference between O/I and $\hat{O}/I$. LPN indicates the logical pattern network, which is implemented in this paper using CNN-LSTM, MLP, Autoencoder, ResNet18, ResNet50, ResNet152 and DCM, respectively. More implementation details about LPN see Sects. 4.1 and 5.

From Formula (9) and Fig. 5, one observes that the LPN merely needs to be provided some training data to automatically learn the logical patterns between a pair of the given images without providing any reasoning patterns beforehand. This is an absolutely data-driven strategy to mine the logical patterns hidden in data.

3.3 Inference form of a LiLi task

Based on the inference form of the DCL 2.2, a LiLi task can be written as the following inference form based on the IF THEN rule.

$$
\begin{array}{ll}
\textit{Antecedent } 1: & \textit{If two input images are } x_1^1 \textit{ and } x_1^2 \textit{ then the output image is } y_1 \\
\textit{Antecedent } 2: & \textit{If two input images are } x_2^1 \textit{ and } x_2^2 \textit{ then the output image is } y_2 \\
\quad\vdots & \quad\vdots \\
\textit{Antecedent } n: & \textit{If two input images are } x_n^1 \textit{ and } x_n^2 \textit{ then the output image is } y_n \\
\textit{Antecedent } n+1: & \textit{If two input images are } x_{n+1}^1 \textit{ and } x_{n+1}^2 \\
\textit{Antecedent } n+2: & \textit{If two input images are } x_{n+2}^1 \textit{ and } x_{n+2}^2 \\
\quad\vdots & \quad\vdots \\
\textit{Antecedent } n+m: & \textit{If two input images are } x_{n+m}^1 \textit{ and} x_{n+m}^2 \\
\hline
\textit{Consequence } n+1: & \textit{The output image is } y_{n+1} \\
\textit{Consequence } n+2: & \textit{The output image is } y_{n+2} \\
\quad\vdots & \quad\vdots \\
\textit{Consequence } n+m: & \textit{The output image is } y_{n+m},
\end{array}
\tag{10}
$$

where x_i^1 and x_i^2 are the input images, y_i is the output image expressing the relation between two input images.

In Formula (10), the n antecedents from 1 to n constituting the training set are used to train the LPN inference model. And the m antecedents from $n+1$ to $n+m$ constituting the testing set are used to test the inference ability of LPN. Based on this, Formula (10) can be further simplified as the following form.

$$
\begin{array}{ll}
\textit{Training antecedent}: & (x_1^1, x_1^2) \longrightarrow y_1 \\
& (x_2^1, x_2^2) \longrightarrow y_2 \\
& \quad\vdots \\
& (x_n^1, x_n^2) \longrightarrow y_n \\
\textit{Testing antecedent}: & (x_{n+1}^1, x_{n+1}^2) \\
& (x_{n+2}^1, x_{n+2}^2) \\
& \quad\vdots \\
& (x_{n+m}^1, x_{n+m}^2) \\
\hline
\textit{Consequence}: & \qquad y_{n+1} \\
& \qquad y_{n+2} \\
& \qquad \vdots \\
& \qquad y_{n+m},
\end{array}
\tag{11}
$$

Formula (11) can be further simplified as the following form by $I_{train} = \{(x_1^1, x_1^2), (x_2^1, x_2^2), \ldots, (x_n^1, x_n^2)\}$, $O_{train} = \{y_1,$ $y_2, \ldots, y_n\}$, $I_{test} = \{(x_{n+1}^1, x_{n+1}^2), (x_{n+2}^1, x_{n+2}^2), \ldots, (x_{n+m}^1, x_{n+m}^2)\}$, and $O_{test} = \{y_{n+1}, y_{n+2}, \ldots, y_{n+m}\}$.

$$
\begin{array}{ll}
\textit{Training antecedent set}: & I_{train} \longrightarrow O_{train} \\
\textit{Testing antecedent set}: & I_{test} \\
\hline
\textit{Consequence set}: & O_{test},
\end{array}
\tag{12}
$$

One can obtain the consequence O_{test} of the antecedent I_{test} by translating three implications $(I_{train} \to O_{train}) \to (I_{test} \to O_{test})$ included by Formula (12) to the following form.

$$
O_{test} = R(I_{train}, O_{train}) \circ I_{test},
\tag{13}
$$

where $R(I_{train}, O_{train}) : [-1, 1]^{2K} \to \{0, 1\}^K$ learned using a data-driven method is a high-dimension mapping function.

According to the above analysis, one can find that on the one hand, the LiLi task has the consistent inference form with the classical propositional calculus, on the other hand they have some different aspects as follows.

- $R_z : [0, 1]^2 \to [0, 1]$ is a duality function. However, $R(I_{train}, O_{train}) : [-1, 1]^{2K} \to \{0, 1\}^K$ is a complex function with high dimensions (K takes 1800 or 900 in this paper).
- R_z needs to be defined beforehand by the experts, while R is learned from a given data set because it is almost

Model	Hyper-parameter
CNN-LSTM	Conv(32,(5,5),l2(1.e-4))->BatchNormalization()->MaxPooling((2,2))->
	Conv(64,(3,3),l2(1.e-4))->BatchNormalization()->MaxPooling((2,2))->
	LSTM(1024, dropout=0.5)
MLP	Dense(256)->Dense(256)->Dense(256)
CNN-MLP	Conv(32,(5,5))->BatchNormalization()->MaxPooling((2,2))->
	Conv(64,(3,3))->BatchNormalization()->MaxPooling((2,2))->
	Dense(4096)
Autoencoder	Conv(32,(5,5))->MaxPooling((2,2))->Conv(64,(5,5))->MaxPooling((2,2))
	Conv(64,(5,5))->UpSampling((2,2))->Conv(32,(5,5))->UpSampling((2,2))
	Cropping2D(((0,1),(0,0)))->Conv(1,(5,5))

Table 1 The hyper-parameter settings on all models

impossible to be defined the function beforehand by human.

In real world, there exist many complex relations that can not be provided beforehand by human beings. When facing this situation, the classical propositional calculus can not work well, even cannot work. Hence, it is desired to design a human-free and data-driven method to learn an unknown relation function. This is the our most main motivation.

4 Experiments

In this section, we compare the performances of several typical deep neural networks on the six LiLi data sets. Next, we detail used models and experimental setup.

4.1 Models and experimental setup

For all models, two images as input are fed into the models, and one image as output is used to compare with the ground truth image. These models are trained to produce one output image in which the correct number is embedded by optimising a mean square error (MSE) loss and using the ADAM or SGD optimiser. The early-stopping is used to choose the optimiser and hyper-parameters of smallest loss estimated on the validation set. In addition, the batch size is set to 32. The hyper-parameter settings and further details on all models see in Table 1. Finally, the performance values are reported on the testing set.

- CNN-LSTM: We develop the model using a standard LSTM module [9]. Since LSTMs are designed to process inputs sequentially, we first pass images sequentially and independently through a 2-layer CNN, and the resulting sequence is handed over to the LSTM. The final hidden state of the LSTM is passed through a fully-connected layer with sigmoid activation function. The model is trained using batch normalization after each convolutional layer and dropout is applied to the LSTM hidden state.
- MLP: The MLP is implemented followed by [15]. The model has three hidden layers each with 256 nodes with ReLU activation functions and one output layer with sigmoid activation. All nodes between adjacent layers are fully-connected.
- CNN-MLP: Inspired by [21], we implement a 2-layer CNN with ReLU activation functions and batch normalizations. The input images are treated as a set of separate greyscale input feature maps for the CNN. The convolutional output is passed through two-layer fully-connected layers, in which the first layer using a ReLU activation

function and the second layer using a sigmoid activation function.
- Autoencoder: A simple autoencoder network is implemented using the idea of [13]. In this model, a 2-layer CNN is used as the encoder network and a 2-layer upsampling network as the decoder network. At last, a convolutional layer is used as the output layer with a sigmoid activation.
- ResNet: We use ResNet architecture as described in [11] and modify the softmax activation function to sigmoid activation function on the last layer of the network. In this paper, we train ResNet-18, ResNet-50 and ResNet-152 on all LiLi data sets and get nearly performances.

4.2 Experiments and analysis on LiLi data sets

In this subsection, we test several typical deep neural networks on these LiLi data sets. Each data set consists of 10,000 training samples, 10,000 validation samples and 20,000 testing samples. The testing samples are not included in the training or validation samples. All models are trained on each training set and stopped when the losses on validation sets no longer decrease. We use an OCR software [41] to recognize the numbers embedded in the predicted images, and then compare them with the ground truth numbers. For one predicted image, it is right when all digits are equal to the ground truth digits. The accuracies of Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication data sets are shown in Table 2.

From Table 2, one observes that all models get the good performances on Bitwise And, Bitwise Or and Bitwise Xor data sets. Only CNN-MLP, Autoencoder and ResNets get the good performances on Addition and Subtraction data sets. Unfortunately, all models fail on Multiplication data set.

The validation loss curves on Bitwise And, Bitwise Or and Bitwise Xor data sets are shown in Fig. 6a–c. Because of the early-stopping, the epochs of these models are different. From these figures, one finds that all models converge to small losses. In addition, the MLP, CNN-MLP and Autoencoder converge faster than the CNN-LSTM and ResNets. The validation loss curves on Addition and Subtraction data sets are shown in Fig. 6d and e. From these figures, one observes that the losses of the CNN-MLP, Autoencoder and ResNets are smaller than other models. Moreover, both of CNN-MLP and Autoencoder converge faster than the ResNets. The validation loss curve on Multiplication data set is shown in Fig. 6f. One can see, from it, that all models have very large losses when they converge.

Next, we try to see if increasing data set size could improve model performances. In this scene, all models are trained on 150,000 training data sets and stopped when
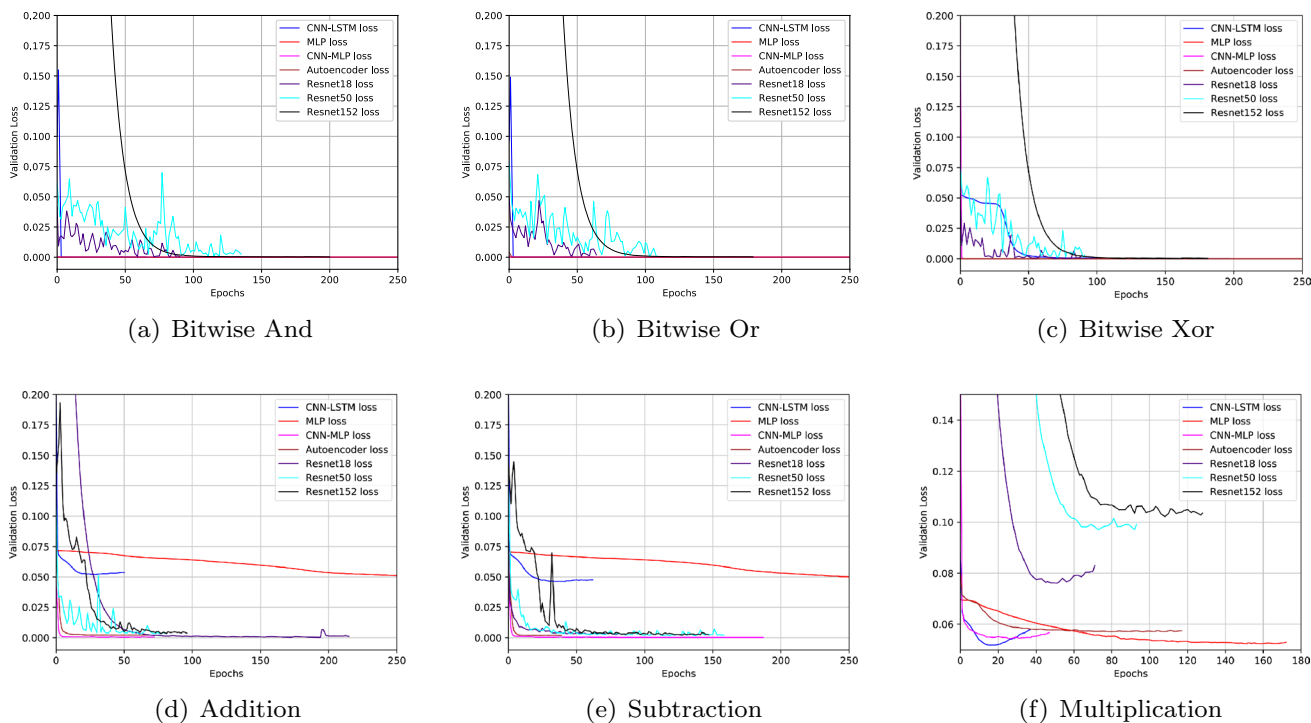
Fig. 6 The validation losses of Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication on 10,000 training data sets

Table 2 The test accuracies of Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication on 10,000 training data sets

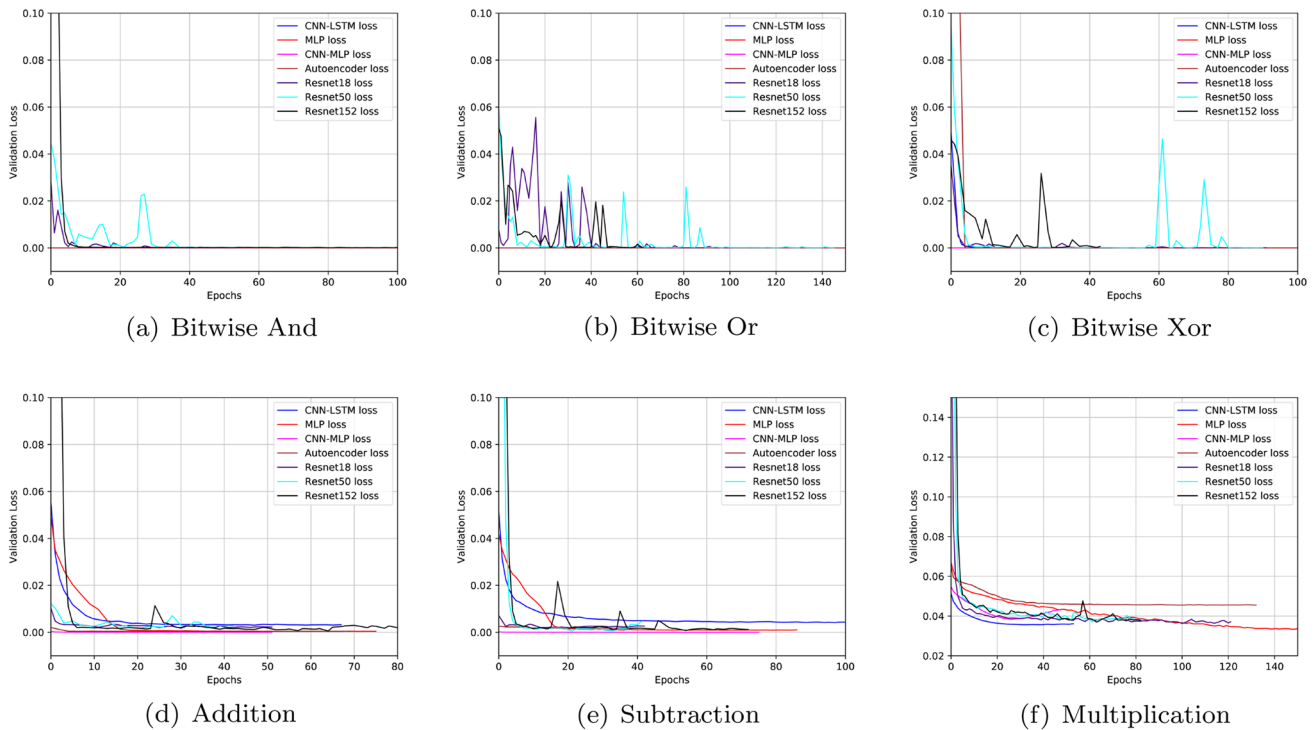| Model	Operations					
	★			★★		★★★
	Bitwise And (%)	Bitwise Or (%)	Bitwise Xor (%)	Addition (%)	Subtraction (%)	Multiplication (%)
CNN-LSTM	100	100	100	0.07	0.38	0.10
MLP	100	100	100	0.21	0.21	0.08
CNN-MLP	100	100	100	96.33	98.69	0.07
Autoencoder	100	100	100	96.78	97.34	0.08
ResNet18	99.96	98.52	99.80	99.86	99.49	0.10
ResNet50	99.92	99.86	99.69	99.14	99.64	0.10
ResNet152	100	100	100	98.74	98.93	0.14

the losses on validation data sets no longer decrease. The accuracies of all models on six LiLi data sets are shown in Table 3.

From Table 3, one observes that most models get the good performances on Bitwise And, Bitwise Or, Bitwise Xor, Addition and Subtraction data sets. It means the performances of models can be improved by increasing the size of data sets. This provides a strategy to solve difficult logic learning problems.

The validation loss curves are shown in Fig. 7. From Fig. 7, one observes that the most of the models converge to smaller losses than before. The validation loss curves on Bitwise And, Bitwise Or and Bitwise Xor data sets are shown in

Fig. 7a–c. From these figures, one finds that the CNN-LSTM and ResNets converge faster than before. The validation loss curves on Addition, Subtraction and Multiplication data sets are shown in Fig. 7d–f, respectively. From Fig. 7d and e, one observes that the losses of all models are smaller than before. But, from Fig. 7f, we observe that all models still have very large losses when they converge on Multiplication data set. A good phenomenon is that the losses of all models are smaller than before.

One guess: the space position plays a significant role in the process of learning logical patterns. It is worth noting that the CNN-LSTM only gets about 80% accuracies on Addition and Subtraction data sets even increasing the size

Fig. 7 The validation losses of Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication on 150,000 training data sets

Table 3 The test accuracies of Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication on 150,000 training data sets

Model	Operations					
	★			★★		★★★
	Bitwise And (%)	Bitwise Or (%)	Bitwise Xor (%)	Addition (%)	Subtraction (%)	Multiplication (%)
CNN-LSTM	100	100	100	84.21	79.22	0.20
MLP	100	100	100	98.79	97.39	0.16
CNN-MLP	100	100	100	99.96	99.96	0.35
Autoencoder	100	100	100	98.17	98.66	0.16
ResNet18	100	100	100	99.50	99.50	0.24
ResNet50	100	100	100	99.56	99.79	0.26
ResNet152	100	100	100	99.98	99.87	0.24

of data sets. However, it get 100% accuracy on Bitwise And, Bitwise Or and Bitwise Xor data sets. The reason is that the CNN-LSTM is fed the input images one by one, learn the features of the images separately so that they almost do not consider the carry or borrow case on addition or subtraction. Each digit of the result of addition and subtraction is affected by the adjacent positions (the influences from carry or borrow), while each digit of the result of bitwise and, bitwise or and bitwise xor is not. If the models want to get high accuracies, they should dispose 2 input images a and b simultaneously on Addition and Subtraction data sets. In order to verify this idea, we develop a model called CNN2-MLP that is similar to CNN-MLP. These two models have same structure and hyper-parameter settings except CNN2-MLP learns features of each of two input images separately. And their structures are shown in Fig. 8.

The validation loss curves of CNN-MLP and CNN2-MLP on the six LiLi data sets are shown in Fig. 9. For Bitwise And, Bitwise Or and Bitwise Xor data sets, both of them converge to the small losses. For Addition and Subtraction data sets, the validation loss of CNN2-MLP is large on 10,000 training data sets. When the size of

Fig. 8 The architectures of CNN-MLP and CNN2-MLP

(a) CNN-MLP

(b) CNN2-MLP

(a) Bitwise And

(b) Bitwise Or

(c) Bitwise Xor

(d) Addition

(e) Subtraction
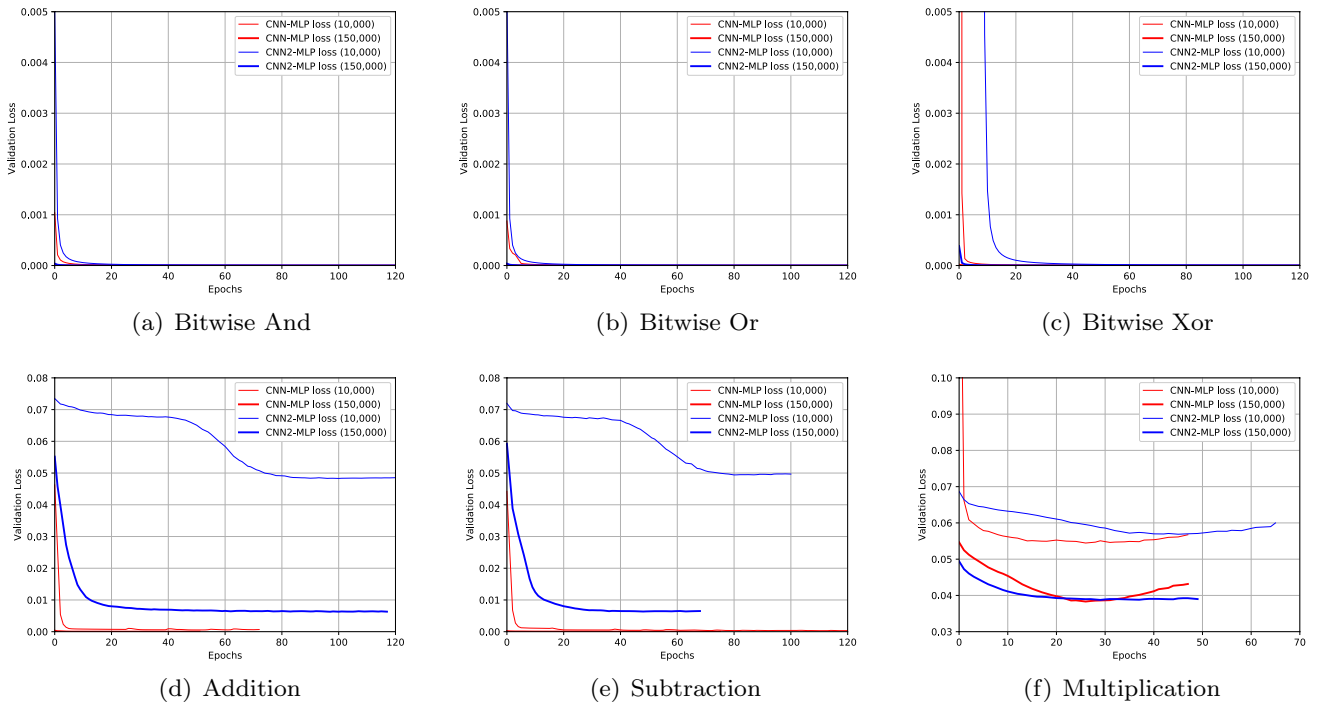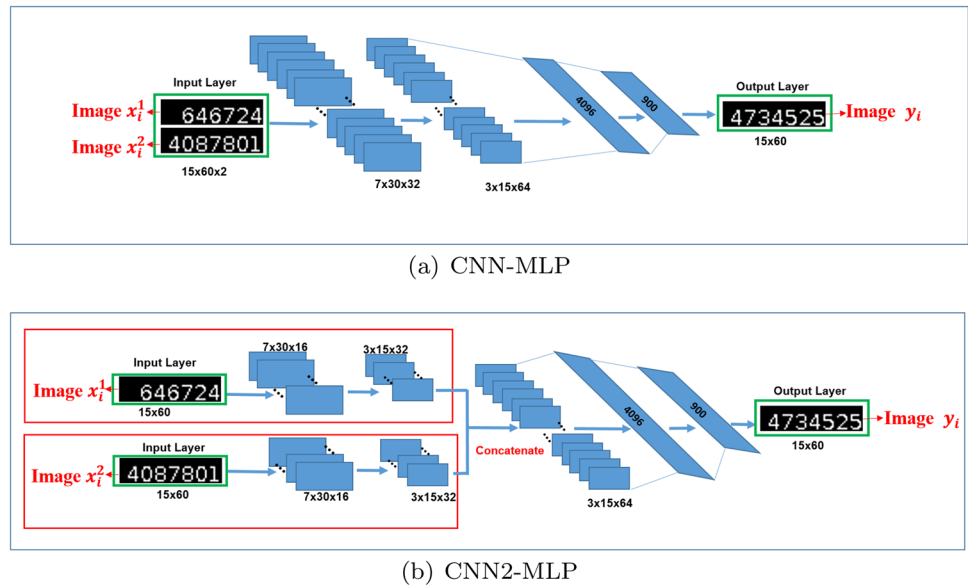
(f) Multiplication

Fig. 9 The validation losses of CNN-MLP and CNN2-MLP on Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication data sets

training data set increasing, the validation loss of CNN2-MLP is smaller than before but still larger than the validation loss of CNN-NLP. For Multiplication data set, both of them converge to the large losses. The test accuracies of CNN2-MLP on Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication data sets are shown in Table 4. CNN2-MLP can not get the good

performances on Addition and Subtraction data sets, but still work well on Bitwise And, Bitwise Or and Bitwise Xor data sets. These experiment results verify that the space position plays a significant role in the process of learning logical patterns.

As the size of the given data increases, the MLP tends to have good performances on Addition and Subtraction

Table 4 The test accuracies of CNN2-MLP on Bitwise And, Bitwise Or, Bitwise Xor, Addition, Subtraction and Multiplication data sets

# training samples	Operations					
	★			★★		★★★
	Bitwise And (%)	Bitwise Or (%)	Bitwise Xor (%)	Addition (%)	Subtraction (%)	Multiplication (%)
150,000	100	100	100	67.47	62.92	0.28
10,000	100	100	100	0.24	0.20	0.05

Fig. 10 The test visual effects of Addition on 10,000 training data set and 150,000 training data set

(a) 10,000 training data set

(b) 150,000 training data set
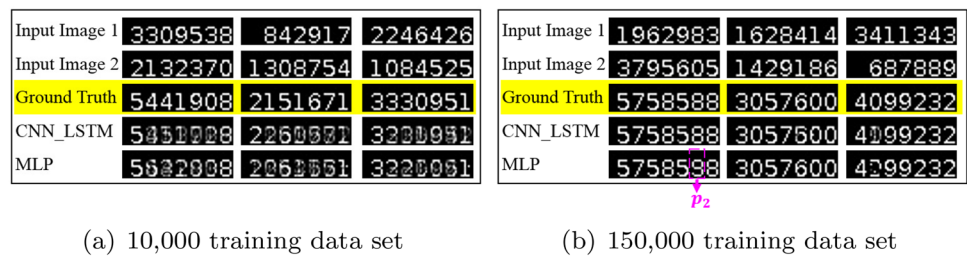
Fig. 11 The test visual effects of Subtraction on 10,000 training data set and 150,000 training data set
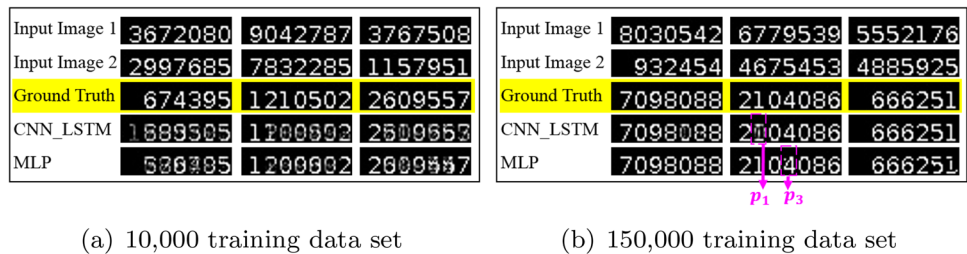
(a) 10,000 training data set

(b) 150,000 training data set

Fig. 12 The test visual effects of Multiplication on 10,000 training data set and 150,000 training data set

(a) 10,000 training data set

(b) 150,000 training data set
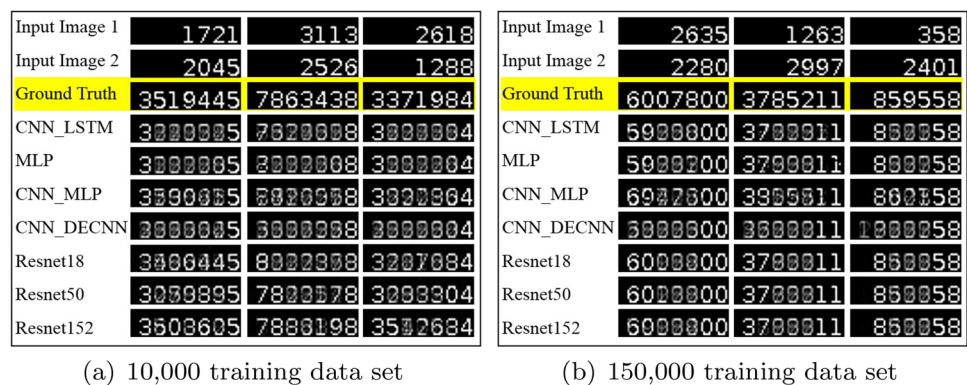
data sets. This is because each digit of the result of the addition and subtraction is affected by the adjacent positions in both input images. In particular, for the MLP, the relation between two images at their arbitrary positions, when data set size is small, it can not focus on the exact relation on their adjacent positions. As soon as the data set gets larger, the defect can be made up.

From what has been discussed above, we can divide these models into three categories:

1. CNN-LSTM: This model is appropriate for this type of task where each digit of the result is only affected by the same position of the input numbers (e.g. Bitwise And, Bitwise Or and Bitwise Xor data sets).
2. MLP: The model is appropriate for this type of task where each digit of the result is affected by all the positions of the input numbers (MLP is more appropriate than other models on Multiplication data sets). If the size of data set is large enough, MLP can focus on the same or adjacent positions of the input numbers (e.g.

Bitwise And, Bitwise Or, Bitwise Xor, Addition and Subtraction data sets).

3. CNN-MLP, Autoencoder and ResNets: These models are appropriate for this type of task where each digit of the result is affected by the same or adjacent positions of the input numbers (e.g. Bitwise And, Bitwise Or, Bitwise Xor, Addition and Subtraction data sets).

Next, from the standpoint of the visual effects, these models are compared. These predicted results output by the models with the poor performances are shown. For Addition and Subtraction data sets, only the CNN-LSTM and MLP get the poor performances; for Multiplication data sets, all models get the poor performances.

For Addition and Subtraction data sets, the visual effects are shown in Figs. 10 and 11. From Figs. 10a and 11a, one observes that most models can clearly learn the first and last digits, and other digits obscurely in output images. As the size of the training data set increases, from Figs. 10b and 11b, one observes that most models can clearly learn most digits in output images. For Multiplication data set, the visual effects are shown in Fig. 12. From Fig. 12a, we observe that most models can only clearly learn the first and last digits and other digits obscurely in output images. As the size of the training data set increases, from Fig. 12b, one sees that most models can clearly learn more digits than before, but still obscurely for most digits in output images. There are many reasons why the performances of the predicted result on the digits is poor. Some predicted digits are very obscure, e.g. the p_1 is shown in Fig. 11b). Some are similar to other digits, e.g. the p_2 is shown in Fig. 10b). Some are right but OCR can not recognize them, e.g. the p_3 is shown in Fig. 11b. Hence the accuracies can be higher in fact.

From above experimental results, one observes that these models can not solve the difficult LiLi task: multiplication. In the next section, an effective solution is provided by dividing this task into a few easier subtasks.

5 Divide and conquer model for Multiplication data set

Although increasing the size of data set has effects on solving the difficult logic learning problems, all models still get the poor performances on Multiplication data set. To our knowledge, many problems are complex and difficult to solve directly, but it becomes easier when decomposed [20, 24, 32, 33, 43]. Artificial algorithm decomposition can effectively reduce the difficulty of learning [2]. Inspired by
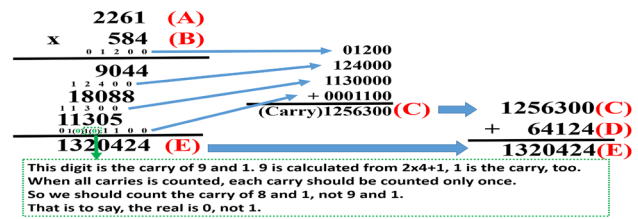

Fig. 13 The procedure of multiplication

this, we propose the DCM to address complex task adopting the decomposition strategy.

We decompose a complex task into k subtasks through the DCM, and the decomposition criterion is that the combination difficulty of subtasks is lower than the complex task.

$$H > f(h_1, h_2, \ldots, h_k), \tag{14}$$

where H is the difficulty of this complex task, h_i is the difficulty of the $i\,th$ subtask, f is the combination difficulty of subtasks and it is determined by all subtasks.

As one sees from Figs. 6f and 7f, the MLP is more robust and can converge to a smaller loss than other models. For multiplication, the value at a given position of E is determined by the values at the given position in A and B and all positions in A and B before that given position. MLP is exactly more appropriate this scene than other models. So we select the MLP as the decomposition module of the DCM.

In this experiment, Multiplication data set is regenerated by adding some information. For training set, each of these samples consists of 4 input images each containing a single integer number. The input images are marked a, b, c and d. The output image marked e is generated by the result of the multiplication operation. The numbers embedded in images a, b, c, d and e are A, B, C, D and E. For testing set, only generate image a, b and e. For per sample, the ranges of A and B are 0–3160. E is the product of A and B. The carry operation occurs when the product of two numbers on one digit is more than ten, and C is used to record the value of carry part, while D is used to record the value of non-carry part. So, the multiplication is divided into the carry part and non-carry part, in other words, the sum of C and D is equal to E. For example, let A and B be "2261" and "584", respectively, and then, C, D and E equal to "1,256,300", "64,124" and "1,320,424", respectively. The calculation procedure is shown in Fig. 13.

The DCM is divided into three subtasks: carry subtask, non-carry subtask and synthetic subtask. First, the carry subtask and non-carry subtask are used to learn the carries of multiplication and multiplication without carry, respectively. And then, the synthetic subtask is used to learn the synthetic pattern of the carry subtask and non-carry subtask.

The network structures of these three subtasks are similar, but the network parameters are different.

1. *Carry subtask* During training, the images a and b are used as the input, image c as the ground-truth result. The network of the carry subtask is fully-connected layers and uses the ReLU as the activation functions in the hidden layers and the sigmoid in the output layer. The carry subtask has 5 hidden layers, and each layer has 256 units.

2. *Non-carry subtask* During training, the images a and b are used as the input, the image d as the ground-truth result. The network of the non-carry subtask is fully-connected layers and uses the ReLU as the activation functions in the hidden layers and the sigmoid in the output layer. The non-carry subtask has 5 hidden layers, and each layer has 256 units.

3. *Synthetic subtask* During training, the images c and d are used as the input, the image e as the ground-truth result. The network of the synthetic subtask is fully-connected layers and uses the ReLU as the activation functions in the hidden layers and the sigmoid in the output layer. The synthetic subtask has 3 hidden layers, and each layer has 256 units.

The ground-truth image is named as x (x can be c, d and e), and the predicted image is named as x'. We hope the number embedded in predicted image e' is equal to the number embedded in ground truth image e, i.e., $E' = E$.

a) *Training* During training procedure, the images a and b are used as the input, e as the ground truth result and e' as the output. It is interesting that the images c and d are both the input and ground truth results. For the carry subtask and non-carry subtask, the images c and d are the ground truth images, however, for the synthetic subtask, the image c and d are the input images. Taking the multiplication formula "$2490 \times 2644 = 6,583,560$"

for example explains the training procedure which is shown in Fig. 14a. A, B, C, D and E are "2490", "2644", "2,575,300", "4,008,260" and "6,583,560", respectively. The carry subtask, non-carry subtask and synthetic subtask are trained separately. For the carry subtask and non-carry subtask, the images a and b are used as the inputs, the images c and d as the ground truth images and the image c' and d' as the outputs, respectively. For the synthetic subtask, the images c and d are used as input, the image e as the ground truth image and image e' as output. The smaller the differences between predicted image c', d' and e' as well as ground-truth image c, d and e are, the better the performance of DCM is.

b) *Testing* In the testing procedure, DCM is an end-to-end model. We take the multiplication formula "$123 \times 124 = 15,252$" for example to explain the testing procedure which is shown in Fig. 14b. A and B are "123" and "124", respectively. In the testing procedure, the DCM only takes images a and b as the inputs, and then directly gets a predicted image e' at the output of the synthetic subtask. Specifically, the inputs are firstly passed through the carry subtask and non-carry subtask to get a carry prediction layer and a non-carry prediction layer, respectively. Then, the two prediction layers are concatenated and passed through the synthetic subtask to get the final prediction result E'. E' is "15,252" and equals to E which shows that the DCM correctly found

Table 5 The test accuracy of each subtask of DCM using 150,000 training examples

Operation	Network branches		
	Carry subtask (%)	Operation without carry subtask (%)	Synthetic subtask (%)
Multiplication	86.25	98.38	84.46

(a) training procedure

(b) testing procedure
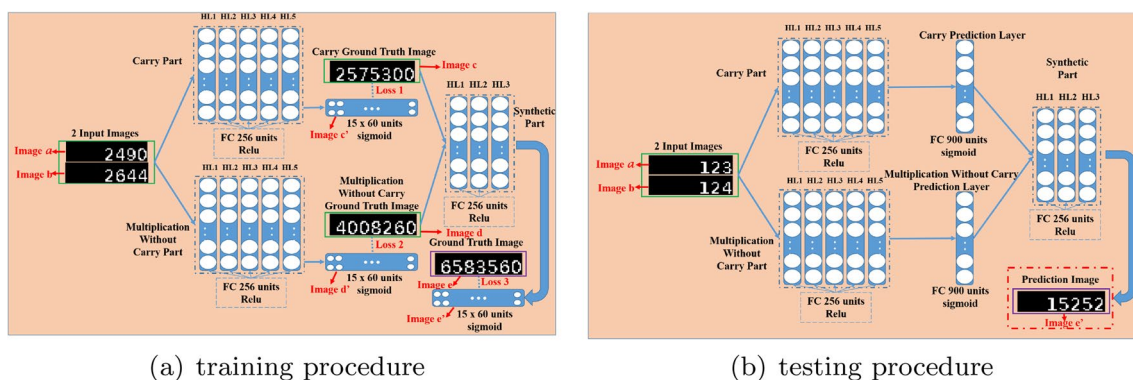
Fig. 14 Training and testing procedure

Fig. 15 The visual effects of Multiplication on 150,000 training set

(a) Both get correct predicted images.

(b) The DCM gets correct predicted images, but MLP does not.

(c) Both get wrong predicted images.

the relation between the images a and b only using the pure visual information.

The DCM is trained using the stochastic gradient descent with momentum 0.9, optimising a mean square error (mse) loss and batch size is fixed 256. The learning rate starts with 0.8, and reduces slowly when the loss plateaus. The training on the carry subtask, non-carry subtask and synthetic subtask terminates when the loss no longer reduces.

The accuracy of each subtask of DCM is shown in Table 5. In contrast, the DCM achieves the surprising accuracy 84.5% which is higher than the MLP on Multiplication data set. Some visual effects from the testing are shown in Fig. 15. In Fig. 15a, both DCM and MLP get correct predicted images. In Fig. 15b, the DCM gets the correct predicted image, but the MLP does not. In Fig. 15c, both DCM and MLP predict wrong images. It can be seen that the last two digits and first two digits in the image of the MLP are predicted correctly, but the rest central 3 digits are uncertain. However, for the DCM, only one digit of the number embedded in the predicted image is uncertain. That is to say, the DCM can confirm more digits than the MLP.

This owns to the special structure of the DCM. DCM divides a complex task into three simple subtasks, carry subtask, non-carry subtask and synthetic subtask, each subtask only learns one aspect of the task. This helps to reduce uncertainty of each predicted digit embedded in the image e'. In order to explain the reason for the effectiveness of the DCM conveniently, we employ some symbols in advance. The goal of the visual logic learning of the arithmetic operations is to compute the value of number 3 in a formula like "number 1 operation number 2 = number 3". We call the digit of number n at the mth position (the rightmost position is 1st) "d_n^m". The complexity of the task is determined by the degree of uncertainty (the amount of possibilities of each digit) in the process of learning logical relation between the input images and output image. For addition, "d_3^m" only has two possibilities, "$(d_1^m + d_2^m)mod10$" or "$(d_1^m + d_2^m + 1)mod10$". The case of "d_3^m" on subtraction is similar to addition. However, the degree of uncertainty of

multiplication is stronger than that of addition and subtraction, where "d_3^m" has ten possibilities.

We assume a formula such as "$d_1^2 d_1^1 \times d_2^2 d_2^1 = d_3^4 d_3^3 d_3^2 d_3^1$" or "$d_1^2 d_1^1 \times d_2^2 d_2^1 = d_3^3 d_3^2 d_3^1$" (if $d_3^4 = 0$). The scope of each digit "d_3^m" (except the digit at rightmost position) is very big, the digit at rightmost position is always an unique and determined value "$(d_1^1 \times d_2^1)mod10$". The DCM can reduce the degree of uncertainty of predicted number 3. For example, "d_3^2" is determined by the carry and non-carry part during multiplication. In the MLP, the scope of "d_3^2" is 0–9, and the scope of the carry at the 2nd position is 0–8. So the carry at the 2nd position is to choose one value in 0–8 out of the range 0–9. The non-carry at the 2nd position is to choose one value in 0–9. So, there are 900 possibilities ($C_{10}^9 C_9^1 C_{10}^1$) for "d_3^2" in fact. In the MLP, "d_3^2" is directly computed. In contrast, our method is first to compute carry and non-carry respectively, and then synthetic these two subtasks. The scope of the carry at the 2nd position is 0–8, so the carry at the 2nd position only needs to determine which one is right in 0–8. The non-carry at the 2nd position is to choose one of 0–9. Hence, there are 90 possibilities ($C_9^1 C_{10}^1$) for "d_3^2". The DCM largely reduces the number of possible values from 900 to 90. Therefore, the DCM confirms more digits than that of the MLP, when the predictions of two models are all wrong.

6 Conclusion

In this study, we have explored an interesting and important research topic: can logic reasoning patterns be directly learned from given data? As a preliminary exploration, the topic has been investigated through a called LiLi task: directly learning logic from a training image set. In this work, many typical neural network models have been used to solve the LiLi task with the good performances on easy and intermediate logic data sets. In order to further solve the difficult task, a new network framework called DCM has been developed using a decompose strategy and adding some label information. This idea also can be applied to other complex logic learning tasks. For example, it is

difficult to compute decimal bit operation directly, we can convert the decimal to binary first, and then compute binary bit operations. The DCM provides a strategy to solve some difficult logic reasoning tasks through combing the domain expert knowledge with data-driven model.

This work is only a preliminary exploration towards learning logic from data. Several issues are worthwhile investigating along this direction, such as mining visual functional relations among multiple variables and directly learning rules from data. These issues are very challenging and meaningful. To this end, more logic reasoning data sets containing complex formulas embedded in the images and more effective models for solving logical reasoning tasks should be specially designed.

Declaration

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Antol S, Agrawal A, Lu J, Mitchell M, Parikh D (2015) VQA: visual question answering. Int J Comput Vis 123(1):4–31
2. Chen L, Huang P, Li Y, Meng Z (2020) Edge-dependent efficient grasp rectangle search in robotic grasp detection. IEEE/ASME Trans Mechatron. https://doi.org/10.1109/TMECH.2020.3048441
3. Chen LC, Papandreou G, Kokkinos I, Murphy K, Yuille AL (2018) Deeplab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. IEEE Trans Pattern Anal Mach Intell 40(4):834–848
4. Chen SM, Cheng SH, Chiou CH (2016) Fuzzy multiattribute group decision making based on intuitionistic fuzzy sets and evidential reasoning methodology. Inf Fusion 27:215–227
5. Colom R, Karama S, Jung RE, Haier RJ (2010) Human intelligence and brain networks. Dialogues Clin Neurosci 12(4):489
6. Dai WZ, Xu Q, Yu Y, Zhou ZH (2019) Bridging machine learning and logical reasoning by abductive learning. In: Advances in neural information processing systems. Vancouver, Canada
7. Golinskapilarek J, Orlowska E (2007) Relational reasoning in formal concept analysis. In: IEEE international fuzzy systems conference. London, UK
8. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in neural information processing systems. Montréal, Canada, pp 2672–2680
9. Graves A (1997) Long short-term memory. Neural Comput 9(8):1735–1780
10. Guo Q, Qian Y, Liang X (2019) Mining logic patterns from visual data. In: International conference on data mining workshops. Beijing, China
11. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: IEEE conference on computer vision and pattern recognition. Las Vegas, USA, pp 770–778
12. He K, Gkioxari G, Dollár P, Girshick R (2017) Mask r-cnn. In: IEEE international conference on computer vision. Venice, Italy, pp 2961–2969
13. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
14. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366
15. Hoshen Y, Peleg S (2016) Visual learning of arithmetic operation. In: Association for the advancement of artificial intelligence. Phoenix, USA, pp 3733–3739
16. Hu R, Andreas J, Rohrbach M, Darrell T, Saenko K (2017) Learning to reason: end-to-end module networks for visual question answering. In: IEEE international conference on computer vision. Venice, Italy, pp 804–813
17. Huang G, Liu Z, Van Der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: IEEE conference on computer vision and pattern recognition. Honolulu, USA, pp 4700–4708
18. Johnson J, Karpathy A, Fei-Fei L (2016) Densecap: fully convolutional localization networks for dense captioning. In: IEEE conference on computer vision and pattern recognition. Las Vegas, USA, pp 4565–4574
19. Johnson J, Hariharan B, Maaten LVD, Li FF, Zitnick CL, Girshick R (2017) CLEVR: a diagnostic dataset for compositional language and elementary visual reasoning. In: IEEE conference on computer vision and pattern recognition. Honolulu, USA, pp 1988–1997
20. Ke L, Zhang Q, Battiti R (2014) Hybridization of decomposition and local search for multiobjective optimization. IEEE Trans Cybern 44(10):1808–1820
21. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436
22. Li M, Chen M, Xu W (2019) Double-quantitative multigranulation decision-theoretic rough fuzzy set model. Int J Mach Learn Cybern 10(5):3225–3244
23. Li SY, Tam LM, Chen HK, Chen CS (2020) A novel-designed fuzzy logic control structure for control of distinct chaotic systems. Int J Mach Learn Cybern 11:2391–2406
24. Liang J, Fadili J, Peyré G (2016) A multi-step inertial forward-backward splitting method for non-convex optimization. In: Advances in neural information processing systems. Barcelona, Spain, pp 4035–4043
25. Liang X, Guo Q, Qian Y, Ding W, Zhang Q (2021) Evolutionary deep fusion method and its application in chemical structure recognition. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2021.3064943
26. Lin Y, Li J, Tan A, Zhang J (2020) Granular matrix-based knowledge reductions of formal fuzzy contexts. Int J Mach Learn Cybern 11:643–656
27. Mizumoto M (1982) Comparison of fuzzy reasoning methods. Fuzzy Sets Syst 8(3):253–283
28. Nilsson NJ (1986) Probabilistic logic. Artif Intell 28(1):71–87
29. Nilsson NJ (1993) Probabilistic logic revisited. Artif Intell 59(1–2):39–42

30. Pearl J (1987) Evidential reasoning using stochastic simulation of causal models. Artif Intell 32(2):245–257

31. Pei DW (2004) On the strict logic foundation of fuzzy reasoning. Soft Comput 8(8):539–545

32. Qian Y, Liang J, Pedrycz W, Dang C (2010a) Positive approximation: an accelerator for attribute reduction in rough set theory. Artif Intell 174:597–618

33. Qian Y, Liang J, Yao Y, Dang C (2010b) Mgrs: a multi-granulation rough set. Inf Sci 180(6):949–970

34. Qian Y, Liang X, Qi W, Liang J, Bing L, Skowron A, Yao Y, Ma J, Dang C (2018) Local rough set: a solution to rough data analysis in big data. Int J Approx Reason 97:38–63

35. Reed S, Akata Z, Yan X, Logeswaran L, Schiele B, Lee H (2016) Generative adversarial text to image synthesis. In: International conference on machine learning. New York, USA, pp 1060–1069

36. Ren S, He K, Girshick R, Sun J (2017) Faster r-cnn: towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell 39(6):1137–1149

37. Shao MW, Lv MM, Li KW, Wang CZ (2020) The construction of attribute (object)-oriented multi-granularity concept lattices. Int J Mach Learn Cybern 11(4):1017–1032

38. She Y, He X, Shi H, Qian Y (2017) A multiple-valued logic approach for multigranulation rough set model. Int J Approx Reason 82:270–284

39. She Y, He X, Qian Y, Xu W, Li J (2018) A quantitative approach to reasoning about incomplete knowledge. Inf Sci 451–452:100–111

40. Shelhamer E, Long J, Darrell T (2017) Fully convolutional networks for semantic segmentation. IEEE Trans Pattern Anal Mach Intell 39(4):640–651

41. Smith R (2007) An overview of the tesseract ocr engine. In: Ninth international conference on document analysis and recognition, vol 2. Curitiba, Brazil, pp 629–633

42. Tadrat J, Boonjing V, Pattaraintakorn P (2012) A new similarity measure in formal concept analysis for case-based reasoning. Expert Syst Appl 39(1):967–972

43. Tan A, Wu WZ, Shia S, Zhao S (2019) Granulation selection and decision making with multigranulation rough set over two universes. Int J Mach Learn Cybern 10(9):2501–2513

44. Tenenbaum JB, Griffiths TL, Kemp C (2006) Theory-based Bayesian models of inductive learning and reasoning. Trends Cogn Sci 10(7):309–318

45. Vinyals O, Toshev A, Bengio S, Erhan D (2016) Show and tell: lessons learned from the 2015 mscoco image captioning challenge. IEEE Trans Pattern Anal Mach Intell 39(4):652–663

46. Wang G (1996) Fuzzy reasoning and fuzzy logic. In: Soft computing in intelligent systems and information processing. Proceedings of the 1996 asian fuzzy systems symposium. Kenting, China, pp 478–483

47. Wille R (1982) Restructuring lattice theory: an approach based on hierarchies of concepts. In: Rival I (ed) Ordered sets. Springer, pp 445–470

48. Wu Q, Shen C, Wang P, Dick A, van den Hengel A (2018) Image captioning and visual question answering based on attributes and external knowledge. IEEE Trans Pattern Anal Mach Intell 40(6):1367–1381

49. Yang Z, Bonsall S, Wang J (2008) Fuzzy rule-based Bayesian reasoning approach for prioritization of failures in FMEA. IEEE Trans Reliab 57(3):517–528

50. Yang Z, He X, Gao J, Deng L, Smola A (2016) Stacked attention networks for image question answering. In: IEEE conference on computer vision and pattern recognition. Las Vegas, USA, pp 21–29

51. Yen J (1999) Fuzzy logic-a modern perspective. IEEE Trans Knowl Data Eng 11(1):153–165

52. Zadeh AL (1973) Outline of a new approach to the analysis of complex systems and decision processes. IEEE Trans Syst Man Cybern SMC 3(1):28–44

53. Zhang P, Goyal Y, Summers-Stay D, Batra D, Parikh D (2016) Yin and yang: balancing and answering binary visual questions. In: IEEE conference on computer vision and pattern recognition. Las Vegas, USA, pp 5014–5022