Extracting Automata from Neural Networks Using Active Learning

Zhiwu Xu, Xiongya Hu, Cheng Wen, and Shengchao Qin

***Abstract*—Deep Learning is a new area of Machine Learning research. Most modern deep learning models are based on an artificial neural network, and benchmarking studies reveal that neural networks have produced results comparable to and in some cases superior to human experts. However, the generated neural networks are typically regarded as incomprehensible black-box models, which not only limits their applications, but also hinders testing and verifying. In this paper, we present an active learning framework to extract automata from neural network classifiers, which can help users to understand the concepts being training or testing the classifiers. In more detail, we use Angluin's L* algorithm as a learner and the neural network under learning as an oracle, employing abstraction interpretation of the neural network for answering membership and equivalence queries. Our abstraction consists of value, symbol and word abstractions. The factors that may affect the abstraction are also discussed in the paper. We have implemented our approach in a prototype. To evaluate it, we have performed the prototype on a MNIST classifier and have identified that the abstraction with interval number 2 and block size 1×28 offers the the best performance in term of *F1 score*. We also have compared our extracted DFA against the DFA learned via the RPNI algorithm and have confirmed that our DFA gives a better performance.**

***Index Terms*—Automata Learning, Neural Network, Active Learning**

I. Introduction

Deep Learning is a new area of Machine Learning research, which has been applied to various fields, including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design and board game programs [1], [2]. Most modern deep learning models are based on an artificial neural network, such as deep neural networks (DNN), deep belief networks (DBN), convolutional neural networks (CNN) and recurrent neural networks (RNN). Benchmarking studies reveal that neural networks have produced results comparable to and in some cases superior to human experts.

However, the generated neural networks are typically regarded as incomprehensible black-box models. They are in practice unlikely to generalise exactly to the concept being trained, and what they eventually learn in actuality is unclear [3]. The opaqueness of neural networks not only limits their applications, but also hinders testing and verifying. Indeed, several lines of work attempt to glimpse into the black-box networks, especially RNN [4]–[11]. They induce rules that mimic the blackbox neural networks as closely as possible, by exploring the possible state vectors of networks, which is often practically impossible at present.

Zhiwu Xu, Xiongya Hu, Cheng Wen and Shenchao Qin are with College of Computer Science and Software Engineering, Shenzhen University, China.
Shengchao Qin is also with School of Computing, Media and the Arts, Teesside University, UK.

Active Learning [12] can learn finite automata (sets of words) precisely from a *minimally adequate teacher*, an oracle capable of answering the so-called *membership* and *equivalence* queries, which has been successfully applied to numerous practical cases in different domains [13]. Recently, Weiss et al. [14] adopted active learning to extract automata from neural networks. But the network systems under learning are RNN acceptors on small regular languages: an input at a time is a symbol, and thus a sequence of inputs is considered as a word. Hence, their approach is not suitable to other neural networks in practice such as CNN, since not all neural networks perform on discrete-time symbol data.

In this paper, we present an active learning framework to extract automata from neural network classifiers, which is inspired by Weiss et al.'s work [14]. But different from their work, we consider each input as a word using abstraction. So the system under learning here can be any neural network. Indeed, we assume that we have no idea about the framework of neural networks: we can not access the state-vectors nor we do not know the relations between two consecutive inputs. For simplicity, we focus on network-acceptors, that is, binary neural network classifiers, since multi-class classifiers can be reduced into several binary classifiers.

Abstraction is the key for scaling model learning methods to realistic application [13]. So the key idea of our approach is to define an abstraction for the neural network classifier under learning. Our abstraction consists of three layers: (1) value abstraction: each value in an input array is mapped into an integer via partitioning; (2) symbol abstraction: a block of multi-dimensional integer array is abstracted as a symbol; and (3) word abstraction: the whole input array is encoded into a word. We also discuss the factors that may affect the abstraction.

Next, we present how to instantiate the active learning framework on neural networks, in particular the membership and equivalence queries [13]. Membership queries can be answered by the neural networks via the word concretization function: we concretise the word that is being queried and then fed the concretised data into the neural network under learning. While equivalence query is more challenge, due to that there are no finite interpretations for neural networks [14]. To address this, we use as an abstract model the automaton that is learned passively from the trained dataset and then perform the equivalence query against the abstract model. If no words that separate the hypothesis and the abstract model are found, then the equivalence query is *yes*. Note that, when a counterexample is found, it may be not that the hypothesis is incorrect, but rather that the abstract model is not precise enough and needs to be refined.

Finally, we have implemented our approach in Java, wherein we use the library *LearnLib* [15] to implement the active learning framework. To evaluate our approach, we conducted a series of experiments on a MNIST classifier. We first test the safety of the MNIST classifier under the abstractions with different interval numbers (*i.e.*, the number of partitioning) and block sizes and have found that it is fine to set the interval number as 2. Secondly, we check whether the abstractions with different interval numbers and block sizes are over-approximated and have identified some suitable block sizes. Thirdly, we conduct some experiments to learn DFAs from the MNIST classifier with some abstractions. The results shows that the abstraction with interval number 2 and block size 1×28 offers the the best performance in term of F1 score. At last, we also conduct the experiments to compare our resulted DFAs against the DFA learned via the RPNI algorithm and the MNIST classifier itself. Although worse than the classifier, our DFA gives a better performance than the RPNI DFA. Nevertheless, there are still some limitations for our approach.

In summary, our contributions are as follows:

- We have proposed an MAT framework to extract automata from neural network, employing abstraction interpretation of the neural network for answering membership and equivalence queries.
- We have conducted several experiments on a MNIST classifier, which demonstrate that our approach is viable, and the resulted DFA has a better performance than the DFA learned via the RPNI algorithm in terms of *F1 score*.

The remainder of this paper is organised as follows. Section II gives the preliminary of DFA and active learning. Section III describes our approach, followed by the experimental results in Section IV. Section V discusses some limitations of our approach. Section VI presents the related work, followed by some concluding remarks in Section VII.

II. Preliminary

In this section, we present some notion about automata [16] and active learning [13].

A. Deterministic Finite Automata

Definition II.1. *A deterministic finite automaton (DFA) is a* 5*-tuple* $(Q, \Sigma, \delta, q_0, F)$*, where*

- Q *is a finite set of* states*,*
- Σ *is a finite set of input symbols and is called the* alphabet*,*
- $\delta : Q \times \Sigma \to Q$ *is the* transition function*,*
- $q_0 \in Q$ *is the* starting state*,*
- $F \subseteq Q$ *is the set of* accepting states.

A *word* or *string* over an alphabet Σ is a finite sequence of symbols from Σ. The *length* of a word is the number of symbols it contains. Note that a word can be empty: the empty word, denoted as ϵ, has length 0 and contains no symbols.

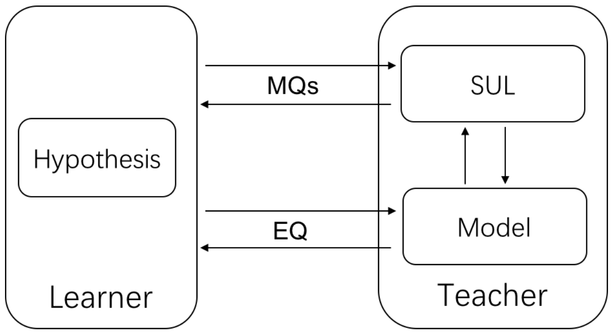

Fig. 1. The MAT Framework

Definition II.2. *Let* $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ *be a DFA and* $w = a_1 a_2 \ldots a_n$ *be a word of length* n *over* Σ*. The automaton* $\mathcal{M}$ *accepts the word* w *if and only if there exists a sequence of states* $r_0, r_1, ..., r_n$ *with the following conditions:*

- $r_0 = q_0$
- $r_{i+1} = \delta(r_i, a_{i+1}), for\ i = 0, ..., n-1$
- $r_n \in F$.

The set of words recognised by a DFA $\mathcal{M}$, called the language of $\mathcal{M}$, is the following set:

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid w \text{ is accepted by } \mathcal{M}\}$$

B. Active Learning Framework

Angluin [12] proposed the first active learning algorithm, the L* algorithm , to learn finite automata from a *minimally adequate teacher* (MAT) in 1987, and today all the most efficient learning algorithms that are being used follow Angluin's approach. In the following, we briefly introduce the MAT framework.

Finite automata (sets of words) can be learned precisely from a *minimally adequate teacher* (MAT), that is, an oracle capable of answering the so-called *membership* and *equivalence* queries:

- membership queries (MQ): the learner asks whether a given word is accepted by the automaton or not, and the teacher answers with the result.
- equivalence queries (EQ): the learner asks whether a given hypothesis $\mathcal{H}$ is equal to the the automaton model $\mathcal{M}$ held by the teacher. The teacher answers yes if this is the case. Otherwise she answers no and supplies a word, the so-called *counterexample*, on which the hypothesis $\mathcal{H}$ and the automaton model $\mathcal{M}$ disagree.

The MAT framework is shown in Figure 1. Initially, the learner knows the static interface of the system under learning (SUL), that is, the sets of input (*i.e.*, multi-dimensional array for neural networks) and output (*i.e.*, *yes* or *no* for recognizers). Then the learner starts to ask a sequence of membership queries and receives the corresponding responses from the teacher. After a "sufficient" number of queries, the learner builds a hypothesis $\mathcal{H}$ from the obtained information, and then sends an equivalence query. If the teacher answers yes, then the hypothesis $\mathcal{H}$ is returned. Otherwise, the learner refines the information with the returned counterexample, and continues on querying.

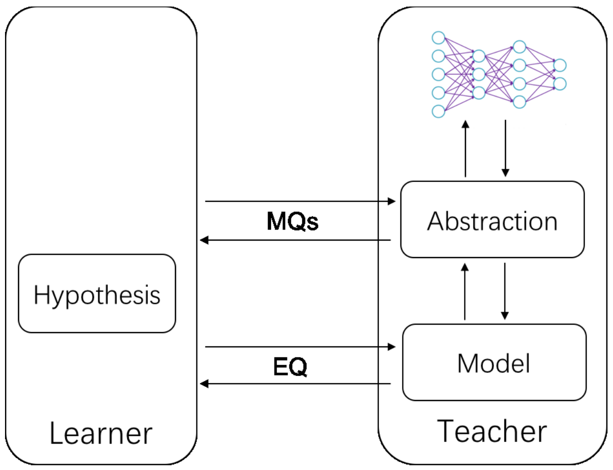

Fig. 2. Framework of Our Approach

III. Approach

In this section, we present an active learning framework to extract automata from neural network classifiers. Our framework is shown in Figure 2, which is a classic MAT framework with an abstraction[1]. To be brief, we make an abstraction between the learner and the system under learning. When queries are sent, the abstraction maps the abstract words into the concrete ones; while the responds are received, the abstraction does the opposite. In the following, we explain how to define an abstraction for neural networks and how to instantiate the active learning framework on neural networks.

A. Abstraction

For simplicity, we focus on network-acceptors, that is, binary neural network classifiers, since we can reduce a multi-class classifier into several binary classifiers. So there are only two outputs for the system under learning, and thus we do not need to abstract them. That is to say, we will abstract only the inputs. Generally, the inputs of neural network classifiers are always multi-dimensional array. As mentioned in Section I, we abstract an input as a word, rather a symbol.

Our abstraction consists of three layers:

- value abstraction: each value in an input array is mapped into an integer via partitioning;
- symbol abstraction: a block of multi-dimensional integer array is abstracted as a symbol;
- word abstraction: the whole input array is encoded into a word.

Value abstraction. Inspired by Omlin and Giles's work [7], we first split the values of the input space into n (equal) intervals, and map each interval into an integer. Formally, let $I_0, \dots, I_{n-1}$ be the intervals and a value d of the input. Then we define a value abstraction function α_v that maps concrete values in the inputs into integers in $\{0, \dots, n-1\}$:

$$\alpha_v(d) = i \quad \text{such that } d \in I_i$$

[1]Some papers use the term *mapper* [13].

When concretizing an abstract integer, we randomly select at most k_v (which can be dependent on the intervals) values to represent the corresponding interval. That is, we define a value concretization function γ_v that maps integers in $\{0, \dots, n-1\}$ into sets of concrete values[2]:

$$\gamma_v(i) = \{d_j \mid d_j \in I_i \text{ and } 0 \le j < k_v\}$$

These two functions can be extended on sets of elements in a natural way.

It is easy to get that $\alpha_v(\gamma_v(i)) = i$ for each integer i. Moreover, if k_v is large enough, we can have $d \in \gamma_v(\alpha_v(d))$ for a given value d. To some extent, (α_v, γ_v) forms a Galois connection [17]. So concerning Galois connections, the larger k_v, the better.

But only Galois connections are not enough here. We also need to consider the safety of neural networks [18]. The composition of the functions α_v and γ_v can be viewed as a manipulation [18]. Performing this manipulation should not result in a different classification. That is, replacing a value d of the input by any value d' obtained by applying this manipulation should not flap the outputs. So we require that d' should be as close to d as possible, that is, n should be as large as possible.

Symbol abstraction. After the value abstraction, each integer can be used as a symbol. But this could yield words that are too long to learn the model. So for scalability, we add a symbol abstraction, which abstracts input arrays into symbols by blocks. For simplicity, in this paper we consider 2-dimensional array with size $iRow \times iCol$. We say a slice of an input array starting from the index (ri, ci) to the index $(ri + oRow, ci + oCol)$ is a block, and the size of the block is $oRow \times oCol$.

A natural way to abstract blocks into symbols is to mapped the blocks into one dimension in row (or column) major order and then encode the one dimension into a base-n number (or a string consisting of the integers in the one dimension). We denote this mapping as α_s^B. By decoding the base-n number (or the string), it is easy to obtain the inverse mapping γ_s^B. It is clear (α_s, γ_s) forms a Galois connection. But a drawback of this solution is that the size of alphabet is $n^{oRow \times oCol}$, which could be too large in practice.

In this paper we use an alternative way to represent a block as its sum. In more detail, we define a symbol abstraction function α_s^S that maps integer blocks of size $oRow \times oCol$ into the sum of the integers in blocks:

$$\alpha_s^S(b) = \sum_{i \in b} i$$

The set of the possible sums of blocks is $\{0, \dots, oRow \times oCol \times (n-1)\}$. So the size of alphabet is $oRow \times oCol \times (n-1) + 1$. Compared to the natural solution, the size of alphabet is quite smaller. But this mapping is not bijective, and thus the inverse mapping does not exist. In order to form Galois connections, similar to value abstraction, we define the inverse mapping from symbols (*i.e.* sums) to sets consisting

[2]In our implementation, we collect sets of values that is mapped into an identity integer from existing data, and select values from the corresponding set when concretizing an abstract integer.

of at most k_s (which can be dependent on the block size and n) blocks whose sum is exact the symbol[3]:

$$\gamma_s^S(sum) = \{b_j \mid \sum_{i \in b_j} i = sum \text{ and } 0 \le j < k_s\}$$

Likewise, these two functions can be lifted to sets of elements in a natural way. Moreover, if k_s is large enough, we can have $b \in \gamma_s^S(\alpha_s^S(b))$ for a given block b. To some extent, (α_s^S, γ_s^S) forms a Galois connection. So concerning Galois connections, the larger k_s, the better.

Let us consider the alphabet size. As discussed above, the alphabet size is linear (exponential resp.) in the block size and the number of intervals n for the abstraction function α_s^S (α_s resp.). So concerning the alphabet size, the smaller the block size and the interval number n, the better.

Due to the mapping is not bijective, the sum abstraction may flap the results of neural networks, . To avoid this, the distance between two blocks of the same sum, or the size of block, should be as small as possible.

Word abstraction. Finally, we split the input array into blocks, and map them into a sequence of symbols (*i.e.*, a word) in row (or column) major order. Algorithm 1 shows the detail of the word abstraction function α_w. The algorithm first invokes the value abstraction α_v to map the values in the input array into integers (Line 1). Then it slides over the integer matrix block by block (Lines $2-10$) and maps each block into a symbol by the symbol abstraction α_s^S (Line 6). Note that here we use a narrow slide on the input array, that is, the blocks to be abstracted are fully contained in the input array. One can use the wide slide with zero-padding as well. In addition, just like the convolution operation of CNN, one can further set the stride sizes for each dimension.

Algorithm 1 Word Abstraction Function $\alpha_w(in)$

```
Input: an input $in$
Output: a word $w$
 1: $in_I = \alpha_v(in)$
 2: $w = \epsilon$, $ri = 0$
 3: while $ri + oRow < iRow$ do
 4:     $ci = 0$
 5:     while $ci + oCol < iCol$ do
 6:         $w = w + \alpha_s^S(in_I[ri : ri + oRow][ci : ci + oCol])$
 7:         $ci = ci + oCol$
 8:     end while
 9:     $ri = ri + oRow$
10: end while
11: return $w$
```

As mentioned above, the symbol abstraction aims to reduce the length of words. According to the word abstraction, we have that the larger the block size, the shorter the word.

The word concretization function γ_w, which is shown in Algorithm 2, does the opposite: it maps a sequence of symbols (*i.e.*, a word) into a sequence of sets of blocks (Lines $5-7$), and combines them into a set of matrices (Lines $8-10$). Note that we require the length of word to be concretized should conform to the size of input (Lines $2-4$). One can release this length condition by zero-padding or discarding the superfluous symbols. But this may break the Galois connections. In addition, one can further encode the sequence of symbols into a final word in a more compact format, such as using run-length encoding (RLE).

[3]In our implementation, we collect sets of blocks that is mapped into an identity symbol from existing data, and select blocks from the corresponding set when concretizing an abstract symbol.

Algorithm 2 Word Concretization Function $\gamma_w(w)$

```
Input: a word $w$
Output: a set $matrix\_set$ of matrices
 1: $rnum = iRow/oRow$, $cnum = iCol/oCol$
 2: if $w.length \neq rnum \times cnum$ then
 3:     return $null$
 4: end if
 5: for $wi = 0, \ldots, w.length - 1$ do
 6:     $data[wi] = \gamma_s^S(word[wi])$
 7: end for
 8: $S = \{m | m[ri : ri + oRow][ci : ci + oCol] \in data[ri \times cnum + ci]\}$
 9: $matrix\_set = \{in | in \in \gamma_v(m)$ and $m \in S\}$
10: return $matrix\_set$
```

If (α_v, γ_v) and (α_s^S, γ_s^S) are Galois connections, then so is (α_w, γ_w). If (α_v, γ_v) and (α_s^S, γ_s^S) does not cause the flapping, then neither does (α_w, γ_w).

Finally, consider the abstraction itself. It should not be an over-approximation. We say a word w is *conflict*, if there exist two inputs of different classifications that are abstracted into w. To avoid over-approximation, the number of the conflict words caused by the abstraction should be as little as possible.

To sum up, to obtain a suitable abstraction (*e.g.*, scalable, safety and non-conflict), one needs to take n, k_v, k_s, and block size into account .

B. Active Learning

In this section, we present how to instantiate the active learning framework on neural networks, in particular the membership and equivalence queries.

Membership query. Membership queries can be answered by the neural networks via the word concretization function. In our abstraction, we map a word into a set of data. As mentioned above, the abstraction may flap the results or yield some conflict words, that is, the classifications of different data in the set of an identity word may not be the same. To address this, we count the numbers of different classifications of the data in the set and take the classification which gets the most votes as the result for the word.

We say a word is *positive* (*negative* resp.) if there are more positive (negative resp.) input arrays that are abstracted into it than the negative (positive resp.) one.

Algorithm 3 gives the procedure of membership query, where NW denote the neural network under learning. Firstly, we concretise the word w that is being queried into a set $matrix\_set$ of possible data, using the word concretization function γ_w (Line 1). If $matrix\_set$ is $null$, that is, the length of word w does not conform to the size of input data, then we return false immediately. Otherwise, we fed each matrix into the neural networks and count the numbers of different classifications (Lines $5-12$). Finally, we return the classification, which gets the most votes (Line 13).

Algorithm 3 Membership Query $MQ(w)$

```
Input: a word $w$
Output: true if $w$ is accepted, otherwise false
 1: $matrix\_set = \gamma_w(w)$
 2: if $matrix\_set == null$ then
 3:     return false
 4: end if
 5: $yes = 0$, $no = 0$
 6: for $matrix\ in\ matrix\_set$ do
 7:     if $NW(matrix)$ then
 8:         $yes++$
 9:     else
10:         $no++$
11:     end if
12: end for
13: return $yes >= no$
```

Equivalence query. As there are no finite interpretations for neural networks [14], equivalence query is more challenge than membership query. To address this, similar to Weiss et al.'s work [14], we use an *abstract representation* of the neural network under learning. But different from Weiss et al.'s work [14], we start with the automaton that is learned passively via the *regular positive and negative inference* (RPNI) algorithm [19] from some test queries, which are selected from the trained dataset. Then we perform the equivalence query against this abstract model. As discussed in [14], when a counterexample is found, it may be not that the hypothesis is incorrect, but rather that the abstract model is not precise enough and needs to be refined.

The procedure[4] of equivalence query is given in Algorithm 4. Firstly, the algorithm tries to find a word that can separate the hypothesis $\mathcal{H}$ and the abstract model $\mathcal{M}$ by the Wp-method test [20] (Line 3). If such a word does not exist, then it returns $null$ (Lines $4-6$), which means the equivalence query is *yes*. Assume a word w is found. Then it checks whether this word is a true counterexample, that is, the classifications of the abstract model and the neural network under learning are the same (Line 7). If it is in that case, then it returns this word as a counterexample to the learner (Line 8). Otherwise, it refines the abstract model with this word (Line 10): it adds the counterexample into the positive set or the negative set dependent on its true classification, and relearns a new automata via RPNI. After that, the algorithm continues on the equivalence query against this refined model.

IV. Experiments

We have implemented our approach in a prototype in Java, wherein we use the library *LearnLib* [15] to implement the MAT learning framework and the RNPI algorithm. To evaluate

[4]In our implementation, we set a bound for the refining time for efficiency, which may yield an incompatible acceptance exception.

Algorithm 4 Equivalence Query $EQ(\mathcal{H}, \mathcal{M})$

```
Input: a hypothesis $\mathcal{H}$ and a model $\mathcal{M}$
Output: a counterexample if $\mathcal{H} \neq \mathcal{M}$, otherwise $null$
 1: while true do
 2:     find a word $w$ that separates $\mathcal{H}$ and $\mathcal{M}$
 3:     if $w$ does not exist then
 4:         return $null$
 5:     end if
 6:     if $\mathcal{M}.isAccepted(w) = MQ(w)$ then
 7:         return $w$
 8:     end if
 9:     refine $\mathcal{M}$ with $(w, MQ(w))$
10: end while
```

our approach, we conduct a series of experiments on a MNIST classifier, which is always used as a *Hello World* program of deep learning. Firstly, we conduct experiments to see the safety of the MNIST classifier under the abstractions with different interval numbers and block sizes. Secondly, we also conduct experiments to test whether the abstractions with different interval numbers and block sizes are over-approximated. Thirdly, we present the experiments to learn DFAs from the MNIST classifier under different abstractions. Fourthly, we also present the comparison of the resulted DFAs against the DFAs learned via the RPNI algorithm and the MNIST classifier itself.

The experiments were conducted on a workstation with Intel Processor i7-7820HQ (2.90GHz) and 32GB memory.

A. MNIST Classifier

The MNIST classifier under learning is a binary classification version of *MnistClassifier* from the tutorial examples of DeepLearning4J [21], which recognises the number 1. It is built on a convolution neural network, which consists of 6 layers, namely, a convolution layer, a pooling layer, another convolution layer, another pooling layer, a dense layer and a output layer. The training dataset and the testing dataset are from the official site [22], wherein each input is 2-dimensional integer matrix with size 28×28.

B. Safety Experiments

As discussed in Section III-A, the interval number n and the block size affect the definition of the abstraction, especially the safety of the neural network under learning. For that, we present in this section some experiments to test the flapping of the MNIST classifier on some selected inputs from the training set under different abstractions with different interval numbers and different block sizes. For simplicity, we use the number of values that are possibly modified by the abstraction manipulation to represent the block size.

In these experiments, for a given interval number n, we randomly select k values from a selected input, and replace each selected value by a random value which shares the same interval with the corresponding selected value. Next, we fed the resulted data into the MNIST classifier and see whether the classifications are flapped. We select 59733 inputs in total from the training set, which are classified correctly by the

TABLE I
FLAPPED RESULTS ON DIFFERENT INTERVAL NUMBERS AND VALUE NUMBERS

n	k	**Flaps**	**Ratio**	n	k	**Flaps**	**Ratio**
2	1	3	0.005%	2	10	16	0.027%
2	100	209	0.350%	2	500	6585	11.024%
3	1	0	0.000%	3	10	6	0.010%
3	100	132	0.221%	3	500	3962	6.633%
5	1	1	0.002%	5	10	4	0.007%
5	100	42	0.703%	5	500	419	7.015%
10	1	1	0.002%	10	10	0	0.000%
10	100	19	0.032%	10	500	93	0.156%

TABLE II
CONFLICT RESULTS ON DIFFERENT BLOCK SIZES

$oRow$	$oCol$	**CPD**	**CND**	**PW**	**CPW**	**NW**	**CNW**
1	28	0	0	6524	0	53114	0
2	28	0	0	6419	0	53114	0
4	28	3	79	3518	77	51740	3
7	28	10	242	4425	240	49393	10
14	28	2530	1334	247	162	5197	516
28	1	0	1	6488	1	53113	0
28	2	0	8	6114	8	53100	0
28	4	80	422	4437	412	51419	79
28	7	2252	1318	1366	483	35128	728
28	14	3121	1198	729	327	4497	832
28	28	3696	1200	37	24	202	73

TABLE III
ALPHABET SIZES AND WORD LENGTHS ON DIFFERENT BLOCK SIZES

n	$oRow$	$oCol$	**dSize**	**Size**	**Length**
2	1	28	21	29	28
2	2	28	41	57	14
2	4	28	48	113	7
2	7	28	126	197	4
2	14	28	143	393	2
2	28	1	21	29	28
2	28	2	41	57	14
2	28	4	81	113	7
2	28	7	124	197	4
2	28	14	146	393	2
2	28	28	244	785	1

MNIST classifier. Table I shows the results, where **Flaps** denotes the number of inputs whose results are flapped by the manipulation, and **Ratio** denotes the percentage of the number of flapped input to the total number of selected inputs.

From the results we can see that, the number of flapped inputs increases as the number of selected values increases. That is to say, the smaller the block size, the less the number of flapped inputs. While as the number of intervals increase, the number of flapped inputs decreases, which indicates that the larger the interval number, the better. These results conform to the discussion in Section III-A. Moreover, the results also shows that when the interval number n is 2 and the number of selected values k is 100 (close to the block size 28×4 or 4×28), the ratio of the flapped inputs is still smaller than 1%. As a small number of intervals yields a small size of alphabet, we suggest to set the interval number n as 2.

C. Conflict Experiments

In this section, we conducted experiments to see whether the abstraction with the given block size is over-approximated, that is, we would like to test how many conflict words that are generated by the abstractions with different block sizes. For that, we perform the abstractions with different block sizes on some selected inputs from the training set, and do a statistic analysis on the abstracted words with respect to their classifications. For simplicity and scalability, we consider the block sizes whose row sizes or column sizes are 28. The test inputs that are selected from the training set is 59733 in total, with 6619 positive inputs and 53114 negative inputs.

The statistic results are given in Table II, where **PW** (**NW** resp.) denotes the number of positive (negative resp.) words, **CPD** (**CND** resp.) denotes the number of positive (negative resp.) inputs that are abstracted into a negative (positive resp.) word and **CPW** (**CNW** resp.) denotes the number of positive (negative resp.) words that have both positive and negative inputs.

The results show that as the block size increases, the number of abstracted words decreases, which conforms to the discussion in Section III-A. Thus it could be more easy to extract the automaton for a larger block size. For example, when taking the whole input as a symbol, there are 239 words in total. But both the number of conflict words and the number of conflict data increase as the block size increases, which indicates that an abstraction with a larger block size is prone to be an over-approximation. In particular, when taking the whole input as a symbol, 55.839% of the positive inputs are abstracted into negative words and 64.865% of the positive words are conflict. Moreover, concerning the conflict words, from the results we can also see that the block sizes of 2×28 and 1×28 perform best, with none conflict data nor words.

In addition, we also count the the alphabet sizes (denoted as **Size**), the number of symbols occurring in the selected inputs (denoted as **dSize**), and the word lengths (denoted as **Length**) on different block sizes, which are shown in Table III. The results show that the larger the block size, the larger the alphabet size and the shorter the word length, which conforms to the discussion in Section III-A. Moreover, we found that the products of the alphabet size and the word length are almost the same. So for scalability, any block size seems fine. But if considering the practical alphabet (*i.e.*, symbols occurring in the inputs), the larger block size could be better.

D. Automata Learning

In this section, we present the experiments to learn DFAs from the MNIST classifier under some abstractions with the interval number $n = 2$, wherein the block sizes are selected based on the experiments above.

To quantitatively validate the models, we use the following performance measures. *Accuracy* is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. *Precision* is the ratio of correctly predicted positive observations to the total predicted positive observations, and *Recall* is the ratio of correctly predicted positive observations to all observations in actual class. *F1 score* is the weighted average of *Precision* and *Recall*, that is, $(2 \cdot Precision \cdot Recall)/(Precision + Recall)$. Intuitively, the higher the measures above, the better the model.

The experimental results are given in Table IV, where **State** denotes the number of states of the resulted DFA (the alphabet size has been given in Table III), the columns **wAcc**, **wPre**, **wRec** and **wF1** respectively denote the *Accuracy*, *Precision*, *Recall* and *F1 score* that are computed with respect to words, and the columns **vAcc**, **vPre**, **vRec** and **vF1** respectively denote the *Accuracy*, *Precision*, *Recall* and *F1 score* that are computed with respect to input values.

Form the results, we can see that the DFAs extracted under the abstraction in rows performs better than the ones under the abstraction in columns both in terms of *F1 score* with respect to words and inputs. For example, the *F1 score* of the block size 1×28 is higher than the one of the block size 28×1. This is because the digit number of 1 is more *regular* in row order than in column order. The results also show that a smaller block size can obtain a higher *F1 score*. For example, the *F1 score* of the block size 1×28 is higher than the one of the block size 2×28 or 28×2. The reason is that a smaller block size can generate a more preciser abstraction. Moreover, concerning the size of extracted DFA, we can extract a larger DFA under a smaller block size. For example, the DFA extracted under the block size 28×1 is the largest one among the results. As discussed before, an input can be abstracted into a longer word under a smaller block size, which thus enlarges the the extracted DFA. In addition, we also perform the abstraction mapping a whole input as a symbol as does in Weiss et al.'s work [14]. Although it has a better performance than the other models on the word layer, the extracted DFA gets the worst performance in the input value layer, due to the abstraction is over-approximated.

E. Comparison

To further evaluate the resulted DFA, we compare it against the DFAs learned via the RPNI algorithm and the MNIST classifier itself. For that, we learn a DFA (denoted as RPNI DFA) via the RPNI algorithm from the training data, wherein all the positive inputs and only one ninth of the negative inputs are selected (to avoid memory overflow). Then we perform all the models on the testing dataset. The results are given in Table V, where both our resulted DFA and the RPNI DFA are extracted under the abstraction with interval number 2 and block size 1×28, and the notations are the same as the ones of Table IV.

Compared to the RPNI DFA, our DFA has a better *Accuracy*, *Precision* and *F1 score*, but a worse *Recall*. This is because that RPNI DFA takes all the positive inputs in the training set into account such that it can recognise more positive inputs in the testing dataset. Although we also use a RPNI DFA as an abstraction model, only part of positive inputs are selected. And during learning, the RPNI DFA is refined with respect to the classifier under learning.

The results also show that our DFA is a little worse than the classifier. A reason for this is that we have set some bounds in our implementation for the learning procedure for efficiency and to avoid memory overflow. Nevertheless, our approach still needs to be improved.

V. Limitations

Although our approach works for the MNIST classifier, there are still some limitations. Firstly, to figure out a suitable abstraction for the neural network under learning is not a easy task. As shown in [18], [23], [24], several deep neural networks, including highly trained and smooth networks optimised for vision tasks, are unstable with respect to so called *adversarial perturbations*. Hence, some neural networks may be too sensitive to the abstraction manipulation to find a reasonable interval number. Even if a reasonable interval number were found, one need to make a compromise between the abstraction and the scalability to find a block size. Moreover, whether a turing machine can simulate a natural neural network is an open question [25]. So in some sense, we cannot define an abstraction without the conflict or the flapping.

Secondly, the scalability is another problem. Generally, the size of inputs of neural networks is in thousands. For such a neural network, either the alphabet may be too large (if a large block size is taken) or the word may be too long (if a small block size is taken) for us to extract the automaton. Taking the MNIST classifier for example, it could last several days for some abstractions to extract the automaton.

Thirdly, our approach is dependent on the dataset. In Section IV, we selected the interval number and the block size via an analysis on the the training dataset. Different datasets may derive different abstractions. To make things worse, it may be the case that an abstraction is suitable for the training dataset, but unsuited for some other testing dataset.

VI. Related Work

In this section, we review some related work. Existing work on DFA extraction from neural networks targets RNNs, which was extensively explored in [26], [27].

Omlin and Giles [7] proposed a global partitioning of the network state space according to q equal intervals along every dimension, and then exploring the network transitions in the partitioned space. Our value abstraction adopts this partitioning, but we work on the input space, instead of the state space.

Cechin [10] presented a approach to extract DFA using k-means and fuzzy clustering. The key idea is to classifier a large sample set of reachable network state using k-means. There are several other work that adopted cluster analysis on state space, including k-means clustering [4], [8], [9], [11] , hierarchical clustering [5], and self-organizing maps [6]. These approaches have to access the state-vectors, while our approach is a block-box one.

Recently, Weiss et al. [14] adopted active learning to extract automata from RNN. Our work is inspired by and similar to this, but different in the follows: (1) we target general neural network, not only RNN; (2) we consider an input is a word, rather than a symbol; (3) we use a DFA that is inferred from some training data as an abstract model for equivalent queries.

VII. Conclusion

In this work, we have proposed a MAT framework to extract automata from neural networks, employing abstraction

TABLE IV
AUTOMATA EXTRACTED UNDER DIFFERENT ABSTRACTIONS

$oRow$	$oCol$	**State**	**wPre**	**wRec**	**wAcc**	**wF1**	**vPre**	**vRec**	**vAcc**	**vF1**
1	28	175	59.284%	85.745%	91.998%	70.100%	59.676%	84.808%	91.850%	70.056%
2	28	96	58.496%	85.216%	91.887%	69.372%	59.270%	84.363%	91.725%	69.624%
28	1	190	49.667%	82.660%	88.993%	62.050%	50.300%	82.106%	88.869%	62.383%
28	2	83	48.317%	82.156%	89.084%	60.848%	50.630%	82.388%	88.989%	62.718%
28	28	3	100.000%	78.378%	96.653%	87.879%	69.649%	40.927%	91.355%	51.557%

TABLE V
COMPARISON AGAINST THE MNIST CLASSIFIER AND THE RPNI DFA

Model	**wPre**	**wRec**	**wAcc**	**wF1**	**vPre**	**vRec**	**vAcc**	**vF1**
Our DFA	55.041%	70.782%	90.176%	61.927%	55.288%	70.988%	90.184%	62.162%
RPNI DFA	31.483%	84.636%	77.476%	45.895%	31.664%	84.744%	77.494%	46.102%
CNN	-	-	-	-	99.69%	99.40%	99.06%	98.63%

interpretation of the neural networks for answering membership and equivalence queries. We have implemented our approach in a prototype and have carried out some interesting experiments on a MNIST classifier. Through experiments, we have found that the DFA extracted from the MNIST classifier under the abstraction with the interval number 2 and the block size 1×28 performs the best. The experimental results have also demonstrated that our resulted DFA has a better performance than the DFA learned via the RPNI algorithm.

As for future work, we may consider a better encoding such as RLE to improve the approach. We can perform experiments on other neural network classifiers. Other models to be extract from neural network are under consideration.

REFERENCES

[1] J. Schmidhuber, "Deep learning in neural networks: an overview," *Neural Netw*, vol. 61, pp. 85–117, 2014.

[2] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[3] C. W. Omlin and C. L. Giles, "Symbolic knowledge representation in recurrent neural networks: insights from theoretical models of computation," *Knowledge-based neurocomputing*, pp. 63–116, 2000.

[4] Z. Zeng, R. Goodman, and P. Smyth, "Learning finite state machines with self-clustering recurrent networks," *Neural Computation*, vol. 5, no. 6, pp. 976–990, 1993.

[5] A. Sanfeliu and R. Alquezar, "Active grammatical inference: A new learning methodology," pp. 191–200, 1995.

[6] P. Tio and J. ajda, "Learning and extracting initial mealy automata with a modular neural network model," *Neural Computation*, vol. 7, no. 4, pp. 822–844, 1995.

[7] C. W. Omlin and C. L. Giles, "Extraction of rules from discrete-time recurrent neural networks," *Neural Networks*, vol. 9, no. 1, pp. 41–52, 1996.

[8] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Representation of finite state automata in recurrent radial basis function networks," *Machine Learning*, vol. 23, no. 1, pp. 5–32, 1996.

[9] M. Gori, M. Maggini, E. Martinelli, and G. Soda, "Inductive inference from noisy examples using the hybrid finite state filter." *IEEE Transactions on Neural Networks*, vol. 9, no. 3, pp. 571–575, 1998.

[10] A. L. Cechin, D. R. P. Simon, and K. Stertz, "State automata extraction from recurrent neural nets using k-means and fuzzy clustering," in *Chilean Computer Science Society, 2003. Sccc 2003. Proceedings. International Conference of the*, 2003, pp. 73–78.

[11] M. Cohen, A. Caciularu, I. Rejwan, and J. Berant, "Inducing regular grammars using recurrent neural networks," *CoRR*, vol. abs/1710.10453, 2017.

[12] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.

[13] F. Vaandrager, "Model learning," *Communications of the Acm*, vol. 60, no. 2, pp. 86–95, 2017.

[14] G. Weiss, Y. Goldberg, and E. Yahav, "Extracting automata from recurrent neural networks using queries and counterexamples," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80. PMLR, 2018, pp. 5247–5256.

[15] F. Howar, M. Isberner, M. Merten, and B. Steffen, *LearnLib Tutorial: From Finite Automata to Register Interface Programs*. Springer Berlin Heidelberg, 2012.

[16] P. Linz, *An Introduction to Formal Languages and Automata*. Jones and Bartlett Publishers, 2001.

[17] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis*. Springer, 1999.

[18] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Computer Aided Verification*. Cham: Springer International Publishing, 2017, pp. 3–29.

[19] J. Oncina and P. García, "Inferring regular languages in polynomial updated time," *Pattern Recognition And Image Analysis*, pp. 49–61, 1992.

[20] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 591–603, 1991.

[21] *Deeplearning4J Examples*, https://github.com/deeplearning4j/dl4j-examples.

[22] *MNIST database*, http://yann.lecun.com/exdb/mnist/.

[23] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. rndi, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Th European Conference on Machine Learning and Knowledge Discovery in Databases*, 2013, pp. 387–402.

[24] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *Computer Science*, 2013.

[25] H. Zenil and F. H. Quiroz, "On the possible computational power of the human mind," *CoRR*, vol. abs/cs/0605065, 2006.

[26] H. Jacobsson, "Rule extraction from recurrent neural networks: Ataxonomy and review," *Neural Computation*, vol. 17, no. 6, pp. 1223–1263, 2005.

[27] Q. Wang, K. Zhang, A. G. O. II, X. Xing, X. Liu, and C. L. Giles, "An empirical evaluation of recurrent neural network rule extraction," *CoRR*, vol. abs/1709.10380, 2017.