CSCI-610 PAPER: DEDUCE AN NFA FROM THE BLACK BOX BY USING MACHINE LEARNING METHOD

A PREPRINT

Zheng Li

December 19, 2021

ABSTRACT

In the present paper, we introduce some related paper that works on reducing the automata structure of the black box by learning its inputs and outputs firstly. By researching these papers, we propose a novel method that can deduce an NFA with structure as simple as possible from the black box. The related program and the ideal is 100% original.

1 Introduction

The learning techniques are usually employed for inferring a formal model such as finite state automaton (FA) or finite state machine (FSM) (also called transducer) of a system whose internal behaviour is implicit and unknown. It has become widespread in the domain of formal verification (e.g., [1, 2, 3, 4]). For instance, if we have permission to use a given software, but cannot view its source code. We can record the outputs of this software by trying a series of different inputs, then construct corresponding automata by using this technique.

However, such general problems are usually extremely difficult[5] so that we need to limit their range. Therefore, in the present paper, we discuss the method that can deduce an NFA with the simplest structure by learning the corresponding dataset. Fig. (1) demonstrates my idea, we can convert this problem to the typical question: **Generate the simplest automata that only accept a given series string.** If we require the automata to be a DFA, the question will be easy. However, such a solution may not be the best, because DFA usually has more finite states than its equivalent NFA. As we know, we can easily convert an NFA to a DFA by computing the closure, but hard to reduce the number of states by converting a DFA to an NFA. Although we can use the Myphill-Nerode Theorem[6] to minimize DFA, we are going to explore the possibility of using machine learning methods.

2 Description of the idea

As we mentioned before, we need to convert the problem into a string recognition problem. Fig.(1) shows the method, we each pair of input and output of the black box, we convert them into a binary vector with fixed length respectively. The map should be a bijection, it implies that different input/output will match different binary vector.

Assuming the input vector in the space $\{0, 1\}^m$ and output vector in the space $\{T, F\}^n$, then when the calculation process of the black box is equivalent to the recognition process of 2^n NFAs(each NFA ouput one component of the output vector). By enumerating all the inputs cases, we will obtain all recognized results, and all the NFAs will be certain. Therefore the example in the dataset consists of two parts, the input binary string and ouput binary vector, the corresponding target is the certained NFAs.

So far we have the dataset, then consider training a model that can output a series of NFAs when inputting a pair of binary string and vector. If the cardinality 2^n is not very large, the Random Forest[7] should be the best model, because it can predict results precisely. If the cardinality 2^n is extremely large, the Deep Neural Network[8] should be the best model, because it can process the high-dimensional features.

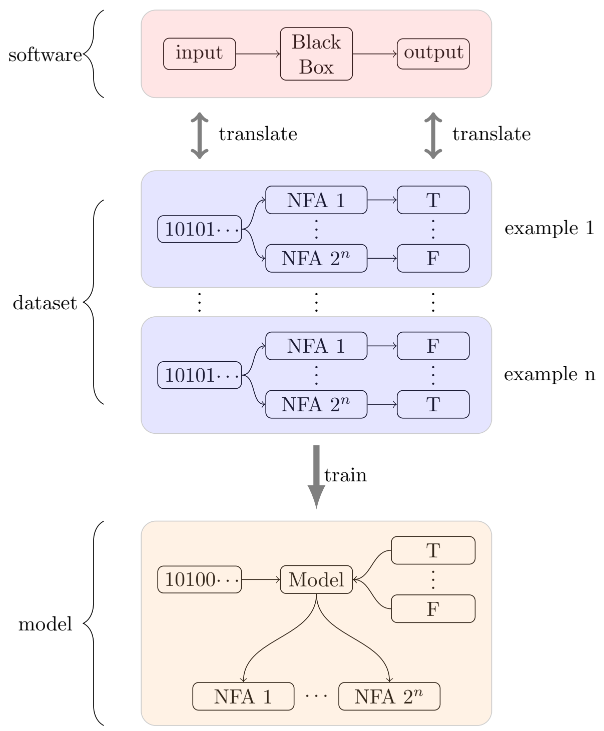

Figure 1: 🟥 Software, 🟦 Examples in dataset, 🟧 Machine learning model,

If using the Deep Neural Network, we can train the model by using the following SGD algorithm, where the $(Q', \Sigma', \delta', q', F') = f(\boldsymbol{x}; \boldsymbol{\theta})$ is the predict NFA and

$$\|f(\boldsymbol{x};\boldsymbol{\theta}) - (Q,\Sigma,\delta,q,F)\| = \|Q-Q'\| + \|\Sigma-\Sigma'\| + \|\delta-\delta'\| + \|q-q'\| + \|F-F'\|$$

Algorithm 1: Train a model to generate NFA

```
Require: Learning rate: $\alpha > 0$.
Require: Loss function $L(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y}) = \|f(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{y}\|$, the combination
         of norms.
Initial parameters $\boldsymbol{\theta}$;
while the loss function $L$ stop decrease do
    Sample a minibatch of $N$ examples from the training set $\{\boldsymbol{x}^{(1)}, \cdots,$
     $\boldsymbol{x}^{(N)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$, such as $\boldsymbol{x}^{(i)} = (101010, T)$,
     $\boldsymbol{y}^{(i)} = (Q,\Sigma,\delta,q,F)$;
    Apply update:
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\frac{1}{N}\sum_{i=1}^{N}\nabla_{\boldsymbol{\theta}}L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),\boldsymbol{y}^{(i)})$;
```

3 Analysis and evaluations

The address of our open-source program is , we list some results of the demo experiment below

string			Accept/Reject vectors					
0	0	0	F	F	F	F	F	⋯
0	0	1	F	F	F	F	F	⋯
0	1	0	F	F	F	F	F	⋯
0	1	1	F	F	F	F	F	⋯
1	0	0	F	F	F	F	F	⋯
1	0	1	F	F	F	F	T	⋯
1	1	0	F	F	T	T	F	⋯
1	1	1	F	T	F	T	F	⋯

For the first instance, the black box read strings ["000", "001", ..., "111"] one by one, then output ["F", "F", "F", "F", "F", "F", "F", "F"] one by one. The model will predict the following NFA

- states: $[q_0, q_1]$
- alphabet: $[0, 1]$
- start state: q_0
- accept states: $\{q_1\}$
- transition function:

	0	1	ε
q_0	$\varnothing$	$\{q_1\}$	$\varnothing$
q_1	$\varnothing$	$\varnothing$	$\varnothing$

For the second instance, the black box read strings ["000", "001", ..., "111"] one by one, then output ["F", "F", "F", "F", "F", "F", "F", "T"] one by one. The model will predict the following NFA

- states: $[q_0, q_1]$
- alphabet: $[0, 1]$
- start state: q_0
- accept states: $\{q_1\}$
- transition function:

	0	1	ε
q_0	$\varnothing$	$\varnothing$	$\{q_1\}$
q_1	$\varnothing$	$\{q_1\}$	$\varnothing$

For the third instance, the black box read strings ["000", "001", ..., "111"] one by one, then output ["F", "F", "F", "F", "F", "F", "T", "F"] one by one. The model will predict the following NFA

- states: $[q_0, q_1]$
- alphabet: $[0, 1]$
- start state: q_0
- accept states: $\{q_0\}$
- transition function:

	0	1	ε
q_0	$\varnothing$	$\{q_1\}$	$\varnothing$
q_1	$\{q_0\}$	$\{q_1\}$	$\varnothing$

For the fourth instance, the black box read strings ["000", "001", ..., "111"] one by one, then output ["F", "F", "F", "F", "F", "F", "T", "T"] one by one. The model will predict the following NFA

- states: $[q_0, q_1]$
- alphabet: $[0, 1]$
- start state: q_0
- accept states: $\{q_0\}$
- transition function:

	0	1	ε
q_0	$\varnothing$	$\{q_1\}$	$\varnothing$
q_1	$\{q_0\}$	$\{q_0, q_1\}$	$\varnothing$

For the fifth instance, the black box read strings ["000", "001", ..., "111"] one by one, then output ["F", "F", "F", "F", "F", "T", "F", "F"] one by one. The model will predict the following NFA

- states: $[q_0, q_1]$
- alphabet: $[0, 1]$
- start state: q_0
- accept states: $\{q_0\}$
- transition function:

	0	1	ε
q_0	$\varnothing$	$\{q_1\}$	$\varnothing$
q_1	$\{q_1\}$	$\{q_0\}$	$\varnothing$

4 Conclusion

From experiments, we are sure the machine learning method works. However, it is not suitable for the case that the string length is extremely large. Why? denote the cardinality of finite states set as n_{states}, the cardinality of alphabet as $n_{alphabet}$ and the length of string as n_{length}, then there are totally $2^{n_{states}} \times (n_{alphabet}^{n_{states}})^{(n_{alphabet}+1)\times n_{states}}$ kinds of NFAs that we have to learn! It's too large! Therefore, if the string length is very large, we recommend using the traditional methods.

References

[1] Doron Peled, Moshe Y Vardi, and Mihalis Yannakakis. Black box checking. In *Formal Methods for Protocol Engineering and Distributed Systems*, pages 225–240. Springer, 1999.

[2] Alex Groce, Doron Peled, and Mihalis Yannakakis. Adaptive model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 357–370. Springer, 2002.

[3] Jamieson M Cobleigh, Dimitra Giannakopoulou, and Corina S Păsăreanu. Learning assumptions for compositional verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346. Springer, 2003.

[4] Sagar Chaki, Edmund Clarke, Natasha Sharygina, and Nishant Sinha. Verification of evolving software via component substitutability analysis. *Formal Methods in System Design*, 32(3):235–266, 2008.

[5] Warawoot Pacharoen, Toshiaki Aoki, Pattarasinee Bhattarakosol, and Athasit Surarerks. Active learning of nondeterministic finite state machines. *Mathematical Problems in Engineering*, 2013, 2013.

[6] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[7] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.