Testing-based Black-box Extraction of Simple Models from RNNs and Transformers

Edi Muškardin[1,2] edi.muskardin@silicon-austria.com and
Martin Tappler[2,1] martin.tappler@ist.tugraz.at and
Bernhard K. Aichernig[2] aichernig@ist.tugraz.at
Silicon Austria Labs, TU Graz-SAL DES Lab[1]*, Graz, Austria and*
Graz University of Technology, Graz, Austria[2]

Editors: François Coste, Faissal Ouardi and Guillaume Rabusseau

Abstract

In this technical report, we outline the testing-based black-box method used to extract simple and interpretable models from RNNs and transformers. Our work was done in the scope of the TAYSIR competition, in which it won the first place.

Keywords: Model extraction, Active automata learning, RNN, Transformers

1. Introduction

Method Overview. We used a testing-oriented black-box extraction method proposed in Muškardin et al. (2022b). It considers a system under learning (SUL), in this case an RNN or a transformer, as a black box and constructs a minimal regular language representation of its input-output behavior. Extraction is performed with active automata learning. This method is suitable for Track 1 without any modifications as both input and output alphabets are discrete, while for Track 2 we add an output mapper which maps a large set of floating-point numbers to a moderately-sized finite discrete set of outputs (explained in mode details in Sect. 3). Approaches from Mayr and Yovine (2018) and Weiss et al. (2018) are similar to ours, given that they also use automata learning to extract models from RNNs, but with different testing strategies. The whole codebase required to extract models from RNNs, as well as learned models, can be found at `https://github.com/emuskardin/taysir_competition_mbt`.

Active Automata Learning. Active automata learning is a method to construct an automaton that conforms to the SUL input-output behavior. This method interacts with the SUL and constructs a behavioral model just by interaction (without knowing anything about the internals of the system). This model is called a hypothesis. A procedure called equivalence query tests whether a hypothesis conforms to the SUL. If it does, the learning procedure stops, and if a counterexample to equivalence is found learning continues by adapting the hypothesis so that the found counterexample is accounted for in its structure. Learning stops when no more counterexamples between the learned model and the SUL can be found. For more details, we refer to Angluin (1987).

Tool Use. For both tracks, we used AALpy from Muškardin et al. (2022a), an automata learning library. It supports a variety of algorithms for deterministic, non-deterministic, and stochastic automata learning. In particular, we used L^* from Angluin (1987) and the KV algorithm from Kearns and Vazirani (1994). Both are active learning algorithms implemented in AALpy.

Difference between learning algorithms. While both L^* and KV yield the same final models if all counterexamples are found, the intermediate process of learning differs. Abstractly, L^* can be seen more as a systematic breath-first exploration of the SUL's behavior, while KV can be seen as a depth-first exploration. L^* returns a final model in fewer learning rounds and it is less dependent on testing, while KV requires n learning rounds for a DFA of size n. This property of KV allows us to terminate learning early in case of models with non-regular behavior that can not be encoded as finite automata.

Model Memory Footprint and Execution Time. All models in Track 1 are encoded as a DFA, while models in Track 2 are encoded as Moore machines. All models are saved as a Python dictionary, which defines the output of each state and all outgoing transitions. Therefore, models have a small memory footprint. For example, a 1000-state DFA with an input alphabet of 10 only uses 37kB. The execution time and memory footprint of these models is minimal, as we only keep track of the current state while iterating over a test sequence.

Precomputation of the Validation Set. For both tracks, we saved output values for all sequences of the given/known validation set. This was done to speed up the testing of the extraction procedure. This step is **optional**, but we have included it in the repository to speed up the reproduction of results. We also used these sequences as additional test cases in equivalence queries.

2. Track 1: Binary Classification

In Track 1, we used the method described in Muškardin et al. (2022b) without any modifications. This method is suited for the extraction of regular languages from RNNs/black-box systems. However, not all RNNs were trained to recognize a regular language. Even if they were trained on a regular language, the RNN's input-output behavior might not be regular due to faults in the RNN's generalization compared with the ground truth.

In the remainder of this section, we outline some findings and specifics of the extraction procedures:

- **Datasets 2,3,4,5,7**: All networks behave as a regular languages, with minimal DFA sizes of {8,9,5,5,2}. Interestingly, the extracted model trained on Dataset 2 that achieves 100% accuracy has 8 states, but when a stronger testing oracle is used we can also extract a model with 10 states.

- **Dataset 6:**: Network trained with Dataset 6 also behaves as a regular language, with 18 states being found without a need for strong testing. These 18 states achieve a 0.0002% error rate on the unknown validation set. However, a strong equivalence oracle can find many counterexamples to this 18-state model, leading to a substantial increase in hypothesis size. We postulate that the underlying RNN was trained on a regular language, but it fails to generalize for all sequences with respect to the ground truth model.

- **Dataset 1**: The network trained with Dataset 1 potentially behaves as a context-free language, as we noticed that a 200-state DFA achieved 0.12% error rate on the unknown validation set, while a 7708-state DFA achieved 0.075% error rate. The 7708-state DFA conforms to all sequences in the known validation set.

- **Dataset 8:** We were unable to identify any meaningful structure in this network. Automata learning would run in a practically infinite loop and early stopping of that loop at various points yielded models that had no improvement in accuracy.
- **Datasets 9, 10, 11:** The behaviour of these networks was not regular, and learning was bounded to 200, 1500, and 500 learning rounds with the KV algorithm. Learned models (regular approximations of a non-regular language) have low error rates (0.007%, 0.014%, 0.007%).

3. Track 2: Density Estimation

In Track 2, we have used the same technique as in Track 1, but with an addition of an output mapper. Once a concrete output value is obtained after executing an input sequence, the applied output mapper maps a virtually unbounded set of output values in the range [0,1] to the final discrete set of abstract outputs.

Mapper Generation. We compute and sort the set of all observed outputs for known validation sequences. This sorted list of outputs is then partitioned into a predefined number of intervals. Each interval acts as a discrete output which is used to represent all values that fall in this interval.

For example, consider that the validation set consists of 9 input sequences, and the obtained outputs are: [0.01, 0.015, 0.02, 0.4, 0.45, 0.5, 0.7, 0.85, 0.95]. Let us set the number of intervals to 3, therefore dividing the sorted output list into 3 equally sized sublists. We then map discrete interval identifiers to the mean of each sublist. We end up with the following mapping: {b1: 0.105, b2: 0.45, b3: 0.83}. These mean values are used to compute the abstract value of a concrete output by choosing the interval with minimal distance to the concrete output. Suppose we are then executing an input sequence on a network and obtain the following outputs: [0.01, 0.3, 0.5, 0.99, 0.2]. This output sequence would be abstracted to [b1, b2, b2, b3, b1].

On the Number of Intervals. For all experiments, the selected number of intervals can be seen at Table 1. A higher number of intervals will yield more accurate models, but with higher automata learning costs to account for a larger output alphabet. Therefore, we set a relatively low number of intervals and focus on the quality of automata learning, as we postulate that hitting the "correct“ interval, i.e., predicting the correct interval with a learned model, is more important than having more intervals. More details on the learning parameterization can be found in the linked repository.

Learning outcome. Given the discrete input alphabet and a discretized output alphabet computed by a mapper, the learning algorithms (L^* or KV) create Moore machines that approximate the SUL's behavior. The output of every state in the learned Moore machine is an interval identifier (eg. b1, b2,..). The extracted model makes predictions by simply tracing an input sequence through the model, and returning a mean value of the reached interval.

Dataset	1	2	3	4	5	6	7	8	9	10
Error Rate (MSE $\times 10^6$)	0.175	0.0097	3×10^{-5}	6×10^{-6}	7×10^{-8}	0.1971	0	0.0443	0	0.1237
Model Size	866	131	110	105	123	318	170	162	55	1412
Learning Rounds	500	100	50	50	50	200	100	100	30	200
# Intervals	200	10	10	10	12	20	15	15	10	20

Table 1: Parameterization and results of model extraction for Track 2.

Results. All extracted models achieve a low mean-square error rate ($\leq 1 \times 10^{-6}$). Results (error rate and number of states in the learned model) and learning parameterization (number of learning rounds and number of intervals) are shown in Table 1.

Acknowledgments

This work has been supported by the "University SAL Labs" initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic based systems.

References

Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75 (2):87–106, 1987. doi: 10.1016/0890-5401(87)90052-6. URL https://doi.org/10.1016/0890-5401(87)90052-6.

Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994. URL https://mitpress.mit.edu/books/introduction-computational-learning-theory.

Franz Mayr and Sergio Yovine. Regular inference on artificial neural networks. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar R. Weippl, editors, *Machine Learning and Knowledge Extraction - Second IFIP TC 5*, volume 11015 of *Lecture Notes in Computer Science*, pages 350–369. Springer, 2018. URL https://doi.org/10.1007/978-3-319-99740-7_25.

Edi Muškardin, Bernhard K. Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. AALpy: an active automata learning library. *Innov. Syst. Softw. Eng.*, 18(3):417–426, 2022a.

Edi Muškardin, Bernhard K. Aichernig, Ingo Pill, and Martin Tappler. Learning finite state models from recurrent neural networks. In Maurice H. ter Beek and Rosemary Monahan, editors, *Integrated Formal Methods - 17th International Conference, IFM 2022*, volume 13274 of *Lecture Notes in Computer Science*, pages 229–248, 2022b.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5244–5253. PMLR, 2018.