ESANN 2024 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 9-11 October 2024, i6doc.com publ., ISBN 978-2-87587-090-2. Available from http://www.i6doc.com/en/.

Recurrent Neural Network based Counter Automata

Sergio Leal Andrés and Luis F. Lago-Fernández *

Departamento de Ingeniería Informática, Universidad Autónoma de Madrid
Ciudad Universitaria de Cantoblanco, 28049 Madrid, Spain

Abstract. This paper presents a neural network architecture that aims to merge RNNs and push-down automata in order to address the recognition of formal languages improving interpretability. The model manages to reproduce a behaviour equivalent to that of an automaton, making it more generalizable and interpretable. Validation has been carried out through several experiments, testing not only convergence but also adaptability and training speed, and comparing the results with similar existing models, as well as with an LSTM. The proposed model serves as a starting point with excellent results, and serves as a basis for future extensions to more sophisticated architectures.

1 Introduction

Recurrent Neural Networks (RNNs) have been extensively studied for their ability to process various formal languages [1, 2]. RNNs, especially those with gating mechanisms like Long-Short Term Memory (LSTM) [3], have demonstrated capabilities in learning both regular and context-free languages to a certain extent [4, 5]. However, training these networks doesn't always result in a set of weights that can effectively recognize certain languages with sufficient generalization. To tackle these issues, many researchers have worked on converting RNNs trained on regular languages into Deterministic Finite Automata (DFA) [6, 7]. A newer method involves making changes to the RNN architecture to make sure it effectively acts as a finite automaton [8]. Here, the network weights represent transition probabilities between the automaton's states, which can be learned by gradient descent. The main benefit of this approach is that the automaton is intentionally constructed by design, rather than being derived from an RNN.

In this work, we aim to present an extension of the model above, which incorporates a differentiable stack in order to deal with non-regular languages. We consider two different stack implementations, one based on the work in [9] and a new approach where several stack situations are managed simultaneously in a probabilistic way. Both models have been implemented considering a stack of just two symbols, which enables the recognition of not only regular languages, but also a subset of the deterministic context-free languages. The innovation of our proposal lies on designing the network architecture to function as a Weighted Finite Automaton (WFA) with stack access, making the model not merely a stack-enhanced RNN, but a true probabilistic push-down automaton. As a proof

*This work has been partially funded by Spanish projects PID2020-114867RB-I00 and PID2023-149669NB-I00 (`http://www.mineco.gob.es/`).

ESANN 2024 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 9-11 October 2024, i6doc.com publ., ISBN 978-2-87587-090-2. Available from http://www.i6doc.com/en/.

of concept, we apply the models to several context-free and regular languages, and compare them to a standard LSTM network.

In the following, we will first describe our model (section 2), the experiments carried out (section 3), and the training settings (section 4). We end with a presentation of the results in section 5 and a brief discussion in section 6.

2 Model

The model adopts a similar approach to the one described in [8], but it includes a differentiable stack inspired by [9]. We consider only two different symbols in the stack alphabet, $\Gamma = \{\epsilon, \kappa\}$, where ϵ is the initial stack symbol and κ is stacked with each push operation. The stack is modeled using a d-dimensional array, $\mathbf{c}^{(t)}$, whose components $c_k^{(t)}$, with $k = 1, ..., d$, represent the probability of the stack containing exactly $k-1$ symbols of type κ. Given a one-hot encoded input vector at time t, $\mathbf{x}^{(t)}$, the previous state probabilities, $\mathbf{q}^{(t-1)}$, and the probabilities for each stack top symbol, $\mathbf{s}^{(t-1)}$, the new state probabilities are obtained by first performing the following tensor operation:

$$z_l^{(t)} = \sum_{i=1}^{|\Sigma|} \sum_{j=1}^{n} \sum_{k=1}^{|\Gamma|} x_i^{(t)} q_j^{(t-1)} s_k^{(t-1)} W_{ijkl}, \tag{1}$$

and then applying the softmax function to $\mathbf{z}^{(t)}$:

$$\mathbf{q}^{(t)} = \sigma(\mathbf{z}^{(t)}). \tag{2}$$

Here W_{ijkl} are the components of a $|\Sigma| \times n \times |\Gamma| \times n$ weight tensor, with n the number of states and Σ the input alphabet. Equivalently, the stack action probabilities, $\mathbf{a}^{(t)}$, are given by:

$$r_l^{(t)} = \sum_{i=1}^{|\Sigma|} \sum_{j=1}^{n} \sum_{k=1}^{|\Gamma|} x_i^{(t)} q_j^{(t-1)} s_k^{(t-1)} V_{ijkl}, \tag{3}$$

$$\mathbf{a}^{(t)} = \sigma(\mathbf{r}^{(t)}), \tag{4}$$

where V_{ijkl} are the components of a $|\Sigma| \times n \times |\Gamma| \times b$ weight tensor and $\mathbf{r}^{(t)}$ and $\mathbf{a}^{(t)}$ are b-dimensional vectors, with b the number of stack actions. We consider $b = 4$, with the four possible stack actions being *push*, *pop*, *stay* and *reset*. The model output at time t is calculated with an additional softmax layer applied to the state probabilities $\mathbf{q}^{(t)}$:

$$y_i^{(t)} = \sigma(\sum_{j=1}^{n} Z_{ij} q_j^{(t)}), \tag{5}$$

where $\mathbf{Z}$ is a $o \times n$ weight matrix, with o the output dimension. Finally, the new stack probabilities, $\mathbf{c}^{(t)}$, are obtained as:

$$c_k^{(t)} = \sum_{i=1}^{b} \sum_{j=1}^{d} a_i^{(t)} c_j^{(t-1)} M_{ijk}, \tag{6}$$

where $\mathbf{M}$ is a fixed tensor whose slices along the first dimension are $d \times d$ matrices that perform the *push*, *pop*, *stay* and *reset* operations, respectively. A second variant of the model uses the stack implementation described in [9].

Following the approach in [8], the weight tensors $\mathbf{W}$ and $\mathbf{V}$ are regularized by adding the following entropy penalization term to the loss function:

$$H(\mathbf{W}, \mathbf{V}) = -\lambda \sum_{i=1}^{|\Sigma|} \sum_{j=1}^{n} \sum_{k=1}^{|\Gamma|} \left(\sum_{l=1}^{n} \tilde{W}_{ijkl} \log \tilde{W}_{ijkl} + \sum_{l=1}^{b} \tilde{V}_{ijkl} \log \tilde{V}_{ijkl} \right), \tag{7}$$

where the tensors $\tilde{\mathbf{W}}$ and $\tilde{\mathbf{V}}$ are the result of applying the softmax operator along the last dimension of $\mathbf{W}$ and $\mathbf{V}$, respectively.

3 Experiments

We perform experiments with the following datasets:

- $a^n b^n$ **grammar**: The first experiment considers strings of the form $a^n b^n$, with $n \geq 0$. It serves as a fundamental example to explain why there are some grammars that cannot be recognized by finite automata. This grammar is key for testing our model because it should not require to alternate between pushes and pops in the stack.
- **Balanced Parenthesis**: In order to keep testing the different models in other context-free grammars, we will test it in the balanced parenthesis problem, which increases the complexity level because nested strings are processed and therefore require alternating between pushes and pops in the stack. The language in focus has two symbols, (and), and it represents strings with balanced parentheses.
- **Tomita grammars**: Apart from the previous context-free grammars, we will evaluate the models with a well-known set of regular languages, known as the Tomita grammars [10], which has become popular in the grammatical inference field.

In all the cases, the training data consist of several strings separated by the \$ symbol. The goal is to predict, for each input symbol, whether the partial string starting in the last \$ is correct or not according to the grammar. The details regarding dataset generation can be consulted in [8] and [11]. We provide examples of input and output strings, for the $a^n b^n$ and the balanced parentheses problems, in table 1.

$a^n b^n$

input	\$	a	a	a	b	b	b	\$	a	a	b	b	\$	a	...
output	1	0	0	0	0	0	1	1	0	0	0	1	1	0	...

balanced parentheses

input	\$	(	)	(	(	)	)	(	)	\$	(	)	(	)	...
output	1	0	1	0	0	0	1	0	1	1	0	1	0	1	...

Table 1: Example of input-output strings for the $a^n b^n$ (top) and the balanced parenthesis (bottom) problems.

4 Model Training

The training phase for all the experiments followed a similar procedure. Initially, we designated the initial state as $\mathbf{q}^{(0)} = (1, 0, 0, ..., 0)$ and the initial stack as empty, setting the model's weights with a random uniform distribution. The model was trained by minimizing the *binary cross-entropy* using an *Adam* optimizer [12], during a maximum of 1000 epochs. The metric used to measure the accuracy was the binary-accuracy. The batch size and sequence length chosen were 32 and 25 respectively. The other relevant hyperparameters were the number of states and the stack size, both of which were set to 20. The regularization parameter is set to $\lambda = 10^{-5}$.

5 Results

We compare three models in terms of their convergence speed and the quality of the solutions they obtain. The first model is the one described in section 2. The second is a variant that uses the stack implementation in [9]. The third one is a standard LSTM with the same number of trainable parameters as the others. We will refer to them as COUNTER, MIKOLOV and, LSTM, respectively.

5.1 Convergence speed

Although the three models are able to solve all the problems with high accuracy, they converge at different speeds. In particular, the COUNTER and the MIKOLOV models show similar convergence rates, whereas the LSTM model converges faster. Fig. 3 (left panel) shows an example for the balanced parentheses problem. However, despite a faster convergence, the LSTM model fails to generalize when long sequences are considered (see section 5.3).

5.2 Convergence to an automaton

In spite of a slower convergence, the COUNTER and the MIKOLOV models converge to deterministic automata which correctly abstract the properties of the languages for all the problems considered, generalizing well to new strings. We show examples of the obtained automata for the $a^n b^n$ problem and the balanced parentheses problem in Fig. 2 and Fig. 1 respectively. The results for the Tomita grammars improve those in [8], making a use of the stack that grants a reduction in the number of states while maintaining the correctness of the solution. The models learn to use the stack as a way to differentiate between several situations in a single state, hence multiplying the effective number of states. This has been observed for all the problems with the exception of the *Tomita 1* language which, due to its simplicity, can be solved with a two-state automaton.

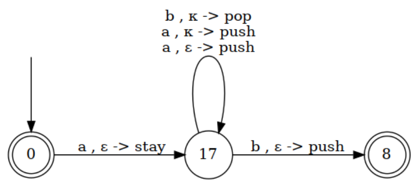

Fig. 1: Automaton obtained for the $a^n b^n$ problem.

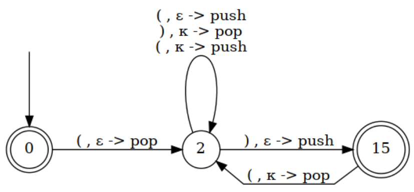

Fig. 2: Automaton obtained for the balanced parentheses problem.

5.3 Adaptability

Here we show how the models adapt when we increase the parenthesis depth in the balanced parentheses problem. The training dataset has been built with a maximum parenthesis depth of 13, whereas several tests of increasing complexity, in terms of their maximum recursive depth are considered. As expected, the COUNTER and the MIKOLOV models maintain a high accuracy when we increase the maximum parentheses depth (Fig. 3, right panel). However, the LSTM model is not able to generalize to very deep strings.

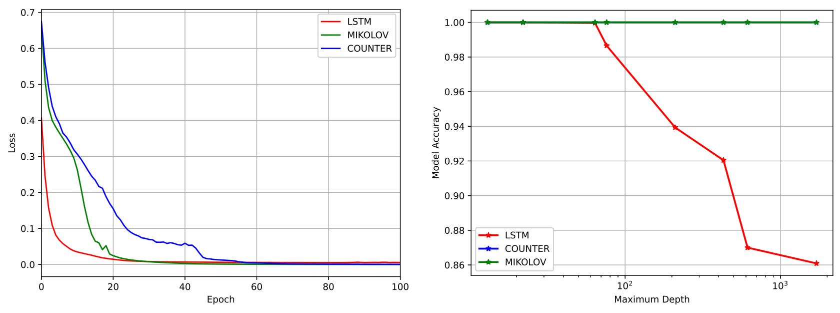

Fig. 3: Left. Loss vs number of training epochs for the balanced parentheses problem. Right. Accuracy on the test set as a function of the maximum parentheses depth (in log scale).

6 Discussion

Recurrent neural networks and automata have been studied separately over the years, but recently novel models propose a combination of these. In this article, we have compared what is probably the most powerful RNN, the LSTM, with some of these mixed models. The results are surprising: we see that more complex structures, which as expected slow down training, produce results with greater adaptability. We have seen how, by design, our neural network behaves as an automaton, which is an example of how to make such intelligent models easier to interpret. The proposed model represents only part of the way forward, which is to explore how even more complex structures, such as trees, full stacks, or even heaps, can help to achieve breakthroughs, bringing RNNs and grammatical inference more strongly together.

References

[1] C. L. Giles, C. B. Miller, D. Chen, G. Sun, H. Chen, and Y. Lee. Extracting and learning an unknown grammar with recurrent neural networks. In *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 317–324, 1991.

[2] Z. Zeng, R.M. Goodman, and P. Smyth. Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990, 1993.

[3] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[4] M. Casey. The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, 8(6):1135–1178, 1996.

[5] F.A. Gers and J. Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Trans. on Neural Networks*, 12(6):1333–1340, 2001.

[6] S. Lawrence, C.L. Giles, and S. Fong. Natural language grammatical inference with recurrent neural networks. *IEEE Trans. Knowl. Data Eng.*, 12(1):126–140, 2000.

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

[8] J. Fdez. del Pozo Romero and L.F. Lago-Fernández. Gradient-based learning of finite automata. In *International Conference on Artificial Neural Networks*, pages 294–305, September 2023.

[9] A. Joulin and T. Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28, [NIPS Conference, Montreal, Quebec, Canada, December 7-12, 2015]*, pages 190–198, 2015.

[10] M. Tomita. Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, pages 105–108, Ann Arbor, Michigan, 1982.

[11] C. Oliva and L.F. Lago-Fernández. On the interpretation of recurrent neural networks as finite state machines. In *Artificial Neural Networks and Machine Learning - ICANN 2019, Munich, Germany, September 17-19, 2019, Proceedings, Part I*, pages 312–323, 2019.

[12] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.