Experiments in Learning Dyck-1 Languages with Recurrent Neural Networks

Nadine El-Naggar, Pranava Madhyastha and Tillman Weyde

City, University of London, United Kingdom

Abstract

Considerable work, both theoretical and empirical, has shown that Recurrent Neural Network (RNN) architectures are capable of learning formal languages under specific conditions. In this study, we investigate the ability of linear and ReLU RNNs to learn Dyck-1 languages in whole sequence classification tasks. We observe that counting bracket sequences is learned but performance on full Dyck-1 recognition is poor. Models for both tasks do not generalise well to longer sequences. We determine correct weights for the given tasks with suitable architectures, but the standard setup for classification surprisingly departs from the correct values. We propose a regression setup with clipping that we find to stabilise correct weights, but it makes learning from random weight initialisation even less effective. Our observations suggest that Dyck-1 languages seem unlikely to be learned by ReLU RNNs for most practical applications.

Keywords

Dyck-1 languages, Formal language learning, Generalisation, Classification, Systematicity

1. Introduction

Neural Networks (NNs) have become the predominant state-of-the-art machine learning technique for natural language processing. However, it is not clear to what extent more systematic tasks, such as counting as needed to parse Dyck-1 languages can be learned by NNs, in particular recurrent ones (RNNs). Weiss et al. [1] and Suzgun et al. [2] show that LSTMs can learn both tasks to some extent under certain conditions. However, simpler RNNs have been less studied for this task, despite theoretical results that they can represent counters and Dyck-1 parsers. In this study, we therefore aim to answer the following research questions:

1. To what extent can RNNs be trained to behave as counters and parse Dyck-1 sequences?
2. Do they learn the required systematic behaviour and generalise to longer sequences?

The contributions of the paper are as follows:

1. We propose two RNN models with specific weights that operate as a linear counter and as a Dyck-1 parser. This is done using linear RNNs and Rectified Linear Unit (ReLU) RNNs.
2. We observe that these models do not learn the correct weights during training.
3. We also find that using a standard classification setup results in correctly initialised models unlearning the correct weights.

HLC'22, September 28–30, 2022

nadine.el-naggar@city.ac.uk (N. El-Naggar); pranava.madhyastha@city.ac.uk (P. Madhyastha); t.e.weyde@city.ac.uk (T. Weyde)

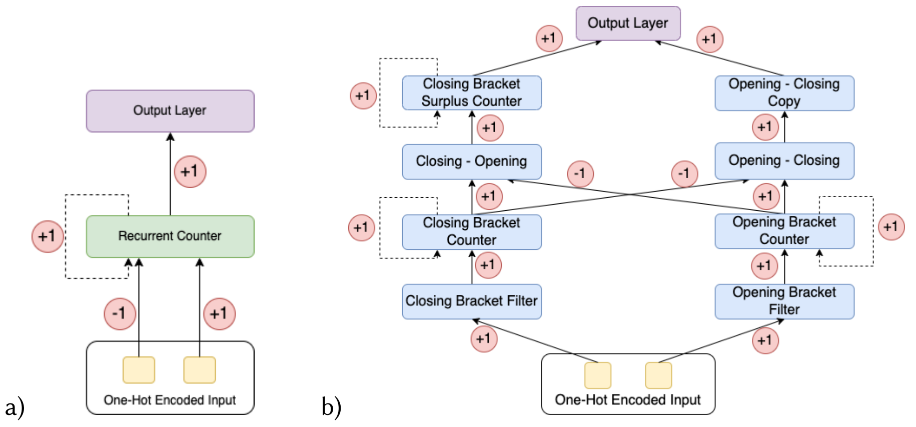

Figure 1: Diagrams of the architecture and correct weights for a) Task 1 (bracket counting) and b) Task 2 (Dyck-1 recognition). Each box represents a neuron with different activation functions indicated by colours (yellow: input, blue: ReLU, green: linear, purple: output - see text for details).

4. We provide evidence that using a regression setup causes models to retain correctly initialised weights, but does not improve their ability to learn the correct weights.

2. Experimental Setup

In our experiments, we evaluate the ability of RNNs to learn Dyck-1 languages in two tasks. Task 1 (bracket counting) detects if there is a surplus of opening brackets in a string, which is not full Dyck-1 recognition. Task 2 (Dyck-1 recognition) detects invalid Dyck-1 sequences, i.e. one that has a surplus of opening brackets overall or that has a surplus of closing brackets at any previous point, versus valid ones. Both tasks require internally counting up for opening and down for closing brackets. For Task 1, calculating the difference and comparing to a threshold is sufficient, which can be achieved with a linear network. For Task 2 we need also to flag negative counter values at any time point for full Dyck-1 recognition. We know that this is possible with a small RNN with ReLU activation from the theoretical results of Siegelmann et al. [3], Leshno et al. [4] and Weiss et al. [1].

Our experimental setup is similar to those of Weiss et al. [1] and Suzgun et al. [2], but there are some important differences. Weiss et al. [1] use $a^n b^n$ sequences and predict the next token. Suzgun et al. [2] only use valid Dyck-1 sequences and determine at every point the valid next tokens, like in Gers and Schmidhuber [5], effectively classifying incomplete versus complete Dyck-1 sequences. While our experiments follow Suzgun et al. [2], our datasets contains also invalid Dyck-1 sequences, i.e. ones with a surplus of closing brackets, which adds complexity to the task. We also use shorter sequences than Weiss et al. [1] and Suzgun et al. [2].

In addition to standard random weight initialisation, we evaluate the effect of training from correct weights. In order to determine the necessary size of the NN and for experimenting with correct weight initialisation, we define two NNs with weights that correctly perform Tasks 1 and 2 as shown in Figure 1. These models operate correctly on sequences of arbitrary length, within the limits of the numeric representation range.

Table 1
Results for Task 1 (bracket counting): average, minimum, and maximum accuracy over 10 runs.

Setup Config/Init.	Train Length	Train Avg(Min/Max)	10 Tokens Avg(Min/Max)	20 Tokens Avg(Min/Max)	50 Tokens Avg(Min/Max)
Class./Random	2	100 (100/100)	69.8 (66.5/78.6)	69.2 (6.04/77.3)	69.0 (66.7/72.7)
Class./Random	4	100 (100/100)	74.8 (74.6/75.7)	94.8 (94.7/95.3)	89.3 (88.7/90.0)
Class./Random	8	100 (100/100)	100 (99.9/100)	96.9 (94.0/100)	92.7 (78.7/98.0)
Reg./Random	2	90.0 (50.0/100)	72.0 (50.0/99.9)	64.5 (50.0/96.0)	61.2 (50.0/91.3)
Reg./Random	4	86.1 (50.0/100)	84.4 (50.0/100)	83.9 (50.0/100)	82.9 (50.0/100)
Reg./Random	8	90.0 (50.0/100)	90.0 (50.0/100)	90.0 (50.0/100)	88.3 (50.0/100)
Class./Correct	2	100 (100/100)	67.9 (67.9/67.9)	71.3 (71.3/71.3)	69.3 (69.3/69.3)
Class./Correct	4	100 (100/100)	74.1 (74.1/74.1)	94.0 (94.0/94.0)	92.0 (92.0/92.0)
Class./Correct	8	100 (100/100)	100 (100/100)	96.7 (96.7/96.7)	93.3 (93.3/93.3)
Reg./Correct	Any	100 (100/100)	100 (100/100)	100 (100/100)	100 (100/100)

Table 2
Results for Task 2 (Dyck-1 recognition): average, minimum, and maximum accuracy over 10 runs.

Setup Config/Init.	Train Length	Train Avg(Min/Max)	10 Tokens Avg(Min/Max)	20 Tokens Avg(Min/Max)	50 Tokens Avg(Min/Max)
Class./Random	2	56.7 (50.0/83.3)	52.7 (50.0/65.3)	54.1 (50.0/71.3)	59.9 (50.0/71.3)
Class./Random	4	55.9 (50.0/79.4)	55.2 (50.0/76.1)	54.5 (50.0/72.7)	54.3 (50.0/71.3)
Class./Random	8	55.3 (50.0/76.6)	55.2 (50.0/76.1)	54.5 (50.0/72.7)	54.3 (50.0/71.3)
Reg./Random	2	56.7 (50.0/83.3)	55.2 (50.0/76.1)	54.5 (50.0/72.7)	54.3 (50.0/71.3)
Reg./Random	4	55.9 (50.0/79.4)	55.2 (50.0/76.1)	54.5 (50.0/72.7)	54.3 (50.0/71.3)
Reg./Random	8	55.3 (50.0/76.6)	56.6 (50.0/83.4)	54.5 (50.0/72.7)	54.3 (50.0/71.3)
Class./Correct	2	100 (100/100)	49.8 (49.8/49.8)	50.0 (50.0/50.0)	50.0 (50.0/50.0)
Class./Correct	4	100 (100/100)	100 (100/100)	86.7 (86.7/86.7)	80.67 (80.7/80.7)
Class./Correct	8	96.7 (96.7/96.7)	97.9 (97.9/97.9)	66.7 (66.7/66.7)	48.0 (48.0/48.0)
Reg./Correct	Any	100 (100/100)	100 (100/100)	100 (100/100)	100 (100/100)

We train both models using sequences of lengths 2, 4, and 8 tokens and test with sequences of 10, 20, and 50 tokens. The datasets contain all possible sequences for lengths 2-10, and a sample of 150 sequences for lengths 20 and 50. All datasets are class balanced. There is a label at the end of each sequence, (Task 1: '1' - incomplete, '0' - balanced or surplus of closing brackets; Task 2: '1' - invalid or incomplete, '0' - valid Dyck-1 sequence). This is different to Gers and Schmidhuber [5], Weiss et al. [1] and Suzgun et al. [2], who use labels at every time step and a target encoding in terms of legal next tokens. We use two different setups, standard classification (sigmoid output activation with cross-entropy loss), and regression setup (clipped output [0, 1] with mean squared error (MSE) loss). We use two initialisation schemes: random weights and correct weights according to Figure 1. We train for 100 epochs using the Adam optimiser by Kingma and Ba [6] and a learning rate of 0.005.

3. Results

In Table 1, we observe that Task 1 is learned on the training set, but generalises less for longer sequences. Longer sequences in the training data do lead to improved generalisation, but still not perfect in most cases. Interestingly, starting from correct weights with the classification setup does not lead to better generalisation. This is intriguing, as the training apparently unlearns the correct weight values. In order to avoid this problem, we developed the regression setup, which does not change the correct weights (last row in Table 1), but it does not learn well from random initialisation. In Task 2, the learning from random weights never leads to perfect training or generalisation. Even with correct initialisation, generalisation is mostly poor.

The models used are custom designed for their respective tasks, and it is possible for these models to be used to solve these tasks with the correct weights. However, in practice, they do not converge to the correct weights. The systematic behaviour to solve these tasks does not emerge in our models during training, irrespective of the setup used, and when the standard classification setup is used, the correctly trained models even unlearn the correct weights. When the regression setup is used, the correctly initialised models do not unlearn the correct weights. However, we observe that the use of the regression setup does not improve the ability of the models to learn systematically and generalise more effectively to longer sequences. This leads us to believe that both of these setups are not suited for this type of task.

4. Conclusion

We have a few interesting observations in our experiments. Learning Dyck-1 sequences from random weights is a difficult task and the results do not generalise to long sequences in any setup not initialised with correct weights. Using longer (and thus more) training sequences leads to some improvements, but generalisation to longer sequences is still limited. Initialising the network with correct weights could help, but even that is not effective in a classification task. The tested approach of using a regression setup (clipping and MSE) can avoid unlearning of correct weights, but it hinders learning from random weight initialisation, and is thus not effective either. Given our results it seems unlikely RNNs would learn Dyck-1 languages in a practical scenario. Using LSTMs could improve the situation, but probably only to a limited extent, given the results by Weiss et al. [1] and Gers and Schmidhuber [5]. Overall, further studies and designs are needed to address reliable learning of Dyck-1 languages with NNs.

The observation is that RNNs fail to learn symbolic patterns and the systematic behaviour required to count brackets and recognise Dyck-1 sequences. This is consistent with studies that focus on the abilities of NNs to learn systematic behaviour, such as Fodor and Pylyshyn [7], Marcus et al. [8], and Lake and Baroni [9]. These studies show that NNs struggle with tasks that are simple for humans to learn from a small number of examples. The difficulty to learn symbolic patterns in a systematic manner is not exclusive to larger models, and is evidently exhibited by our very small models. Achieving systematicity in smaller models can potentially serve as a stepping stone towards achieving systematicity for larger models. In the future we aim to develop a deeper understanding of the learning dynamics of our models, and develop methods that can improve systematic learning.

References

[1] G. Weiss, Y. Goldberg, E. Yahav, On the practical computational power of finite precision rnns for language recognition, in: I. Gurevych, Y. Miyao (Eds.), Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 2: Short Papers, Association for Computational Linguistics, 2018, pp. 740–745. URL: https://aclanthology.org/P18-2117/. doi:10.18653/v1/P18-2117.

[2] M. Suzgun, S. Gehrmann, Y. Belinkov, S. M. Shieber, LSTM networks can perform dynamic counting, CoRR abs/1906.03648 (2019). URL: http://arxiv.org/abs/1906.03648. arXiv:1906.03648.

[3] H. T. Siegelmann, B. G. Horne, C. L. Giles, Computational capabilities of recurrent narx neural networks, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 27 (1997) 208–215.

[4] M. Leshno, V. Y. Lin, A. Pinkus, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural networks 6 (1993) 861–867.

[5] F. A. Gers, J. Schmidhuber, LSTM recurrent networks learn simple context-free and context-sensitive languages, IEEE Trans. Neural Networks 12 (2001) 1333–1340. URL: https://doi.org/10.1109/72.963769. doi:10.1109/72.963769.

[6] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015. URL: http://arxiv.org/abs/1412.6980.

[7] J. A. Fodor, Z. W. Pylyshyn, Connectionism and cognitive architecture: A critical analysis, Cognition 28 (1988) 3–71.

[8] G. F. Marcus, S. Vijayan, S. B. Rao, P. M. Vishton, Rule learning by seven-month-old infants, Science 283 (1999) 77–80.

[9] B. M. Lake, M. Baroni, Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks, in: J. G. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of *Proceedings of Machine Learning Research*, PMLR, 2018, pp. 2879–2888. URL: http://proceedings.mlr.press/v80/lake18a.html.