2018

Using Regular Languages to Explore the Representational Capacity of Recurrent Neural Architectures

Abhijit Mahalunkar
Tecnological University Dublin, abhijit.mahalunkar@tudublin.ie

John D. Kelleher
Technological University Dublin, john.d.kelleher@tudublin.ie

Recommended Citation

Mahalunkar, A. and Kelleher, J.D. (2018). Using Regular Languages to Explore the Representational Capacity of Recurrent Neural Architectures. *27th International Conference on Artificial Neural Networks, ICANN 2018; Rhodes; Greece; 4 October 2018 through 7 October*. doi:10.1007/978-3-030-01424-7_19

Using Regular Languages to Explore the Representational Capacity of Recurrent Neural Architectures

Abhijit Mahalunkar[0000−0001−5795−8728] and
John D. Kelleher[0000−0001−6462−3248]

Dublin Institute of Technology
Ireland
abhijit.mahalunkar@mydit.ie,john.d.kelleher@dit.ie

Abstract. The presence of Long Distance Dependencies (LDDs) in sequential data poses significant challenges for computational models. Various recurrent neural architectures have been designed to mitigate this issue. In order to test these state-of-the-art architectures, there is growing need for rich benchmarking datasets. However, one of the drawbacks of existing datasets is the lack of experimental control with regards to the presence and/or degree of LDDs. This lack of control limits the analysis of model performance in relation to the specific challenge posed by LDDs. One way to address this is to use synthetic data having the properties of subregular languages. The degree of LDDs within the generated data can be controlled through the k parameter, length of the generated strings, and by choosing appropriate *forbidden strings*. In this paper, we explore the capacity of different RNN extensions to model LDDs, by evaluating these models on a sequence of SPk synthesized datasets, where each subsequent dataset exhibits a longer degree of LDD. Even though SPk are simple languages, the presence of LDDs does have significant impact on the performance of recurrent neural architectures, thus making them prime candidate in benchmarking tasks.

Keywords: Sequential Models, Long Distance Dependency, Recurrent Neural Networks, Regular Languages, Strictly Piecewise Languages.

1 Introduction

A Recurrent Neural Network (RNN) is able to model temporal data efficiently [1]. In theory, RNNs are capable of modeling infinitely long dependencies. A long distance dependency (LDD) describes a contingency (or interaction) between two (or more) elements in a sequence that are separated by an arbitrary number of positions. LDDs often occur in natural language, for example in English there is a requirement for subjects and verbs to agree, compare: "*The **dog** in that house **is** aggressive*" with "*The **dogs** in that house **are** aggressive*". However, in practice successfully training an RNN to model LDDs is still extremely difficult,

due in-part to exploding or vanishing gradients [2,3]. There have been significant advances in this domain, and various architectures have been developed to address the issue of LDDs [4,5,6,7,8,9,10,11]. Indeed, the fact that a number of RNN extensions are specifically designed to address the problem of modeling LDDs is a testament to the fundamental importance of the challenge posed by LDDs.

In order to test the representational capacity of these models and aide in future development of new models, there is a growing need for large datasets which manifest various degrees of LDDs. Various benchmarking datasets and tasks which exhibit such properties are currently being employed [12,7,13,4]. However, using them provides no experimental control over the degree of LDD these datasets exhibit. Although, the copy and add task [4] does have control over this factor, the dataset generated via this scheme does not possess comparable complexity with datasets sampled from real world sequential processes.

Strictly k-Piecewise (SPk) languages, as studied by Rogers et al. [14], are proper subclasses of piecewise testable languages [15]. SPk languages are natural and can express some of the kinds of LDDs found in natural languages [16,18]. In relation to research on LDDs, SPk languages are particularly interesting because by controlling the parameter k and the length of the strings, one can control the maximum LDD in the dataset, and by choosing appropriate *forbidden strings*, it is possible to simulate a natural dataset exhibiting a certain degree of LDD. These properties make SPk languages prime candidate for benchmarking tasks.

Contribution: This research used a finite-state implementation of an SP*2* grammar to generate strings of varying length, from 2 to 500. SP*2* is analogous to subject-verb agreement in English language, thus using this grammar generates LDDs of similar complexity, and controlling the length of the strings generated controls the maximum LDD span in the dataset. Appropriate *forbidden strings* were chosen. State-of-the-art sequential data models were trained to predict the next character for every generated dataset. It was observed that as the length of the strings in the datasets increased the perplexity of the models increased. This is due in-part to the limitations of the representational capacity of these models. However, of the models tested it was observed that Recurrent Highway Networks display the lowest perplexity on character prediction task for large sequences exhibiting very high LDDs.

2 Recurrent Neural Architectures for LDDs

The focus of this paper is to experimentally evaluate the ability of modern RNN architectures to model LDDs by testing current state-of-the-art models on datasets of SPk sequences which exhibit LDDs of varying lengths. For our experiments we chose the following architectures as the relevant representatives of RNNs: Long Short Term Memory [4], Recurrent Highway Networks [9] and Orthogonal RNNs [10]. This choice of networks was based on the fact that (a) each of these networks were specifically designed to address performance issue of the standard RNN while modeling LDD datasets, and (b) taken together the

set of selected models provide coverage of the different approaches found in the literature to the problem of LDDs.

LSTMs were an early effort in addressing the vanishing gradient effect by introducing *"constant error carousels"*, which enforced *constant error flow through* thereby bridging minimal time lags in excess of 1000 discrete steps. Neural Turing Machines are memory augmented networks. They are composed of a network *controller* and a memory bank. These components allowed the network to provide attention to different memory locations. Recurrent Highway Networks (RHNs) extended the LSTM architecture to allow step-to-step transition depths larger than one. Orthogonal RNNs (ORNNs) extend the standard RNN architecture by enforcing soft or hard orthogonality on the weight matrix.

3 Benchmarking Datasets

There is a relatively small number of datasets that are popular for testing the representational capacity of RNNs. Most of these datasets are known to exhibit LDDs, which is a necessary criteria for their selection as a benchmarking dataset. The *Penn Treebank* [12] (PTB) is one of these datasets. It consists of over 4.5 million words of American English and was constructed by sampling English sentences from a range of sources. The *WikiText* language modeling dataset [7] was released in 2016 and has become a popular choice for language modeling experiments. It is a collection of over 100 million tokens extracted from various Wikipedia articles. This dataset is much larger than the PTB, which is the primary reason that it is preferred to the PTB in recent works. Although, the PTB and WikiText differ in terms of the sources that the sentences they contain are sampled from, both dataset exclusively contain English language sentences. Hence both the datasets are constrained by English language grammar, and therefore will exhibit similar LDD characteristics. Moreover, it is unclear what these LDDs are because the data is sampled from a natural process (the English language) the LDD characteristics of which are not accurately estimated.

The difficulty of using naturally occurring datasets to investigate LDDs has been recognized and several synthetic benchmarks have been used to test the ability of RNNs to learn LDDs in sequential data. The copy and adding tasks, introduced in [4], is one such example. The task entails remembering an input sequence followed by a string of blank inputs. The sequence is terminated using a delimiter after which the network must produce the input sequence, ignoring the string of blanks inputs that follow the original sequence [10]. This task provides an experimenter with a great degree of control over the length of LDD in the dataset they synthesize in order to train and test their models.

Another method of testing models on simulated LDDs, is to train them to learn the MNIST image classes [13]. This is achieved by sequentially feeding all the 784 pixels of a MNIST image to the model under test and then training the network to classify MNIST image category. Every image is fed to the network pixel by pixel, starting from the top left pixel and finishing at the bottom right

pixel. This simulates LDDs of length 784 as the network has to remember all the 784 pixels in order to classify the images.

Formal languages, have previously been used to train RNNs and investigate their inner workings. The *Reber grammar* [19] was used to train various first order RNNs [21,22]. The Reber grammar was also used as a benchmarking dataset in the original work on LSTM models [4]. Regular languages, studied by Tomita [20], were used to train RNNs to learn grammatical structures of the string. A very recent example of research using formal languages to evaluate RNNs is Avcu et. al. [17]. The work presented in this paper falls within this tradition of analysis, however it extends the previous research on using formal languages by: (a) broadening the variety of LDDs within the generated datasets, (b) evaluating a broader variety of models, and (c) using language model perplexity as the evaluation metric.

4 Formal Language Theory and Regular Languages

Formal Language Theory (FLT) finds its use in various domains of science. Primarily developed to study the computational basis of human language, FLT is now being used to extensively analyze any rule-governed system [23,24,25]. Regular languages are the simplest grammars (type-3 grammars) within the Chomsky hierarchy which are driven by regular expressions. Subregular languages, e.g. Strictly k-Piecewise or Strictly k-Local, are subclasses of regular languages. These languages can be identified by mechanisms much less complicated than Finite-State Automata. Many aspects of human language such as local and non local dependencies are similar to subregular languages [26], and there are certain types of LDDs in human language which allow finite-state characterization [18]. These types of LDD can be modeled using Strictly k-Piecewise languages.

4.1 Strictly Piecewise Languages

In order to explain how we used SPk languages to generate datasets appropriate to our experimental goals it is first necessary to present an explanation of these languages. Following [17,16,14], a language L is described by a finite set of symbols, i.e. an alphabet, denoted by Σ. The symbols are analogous to words or characters in English, music notes in music theory, genes in genomics, etc. A set Σ^* is a *free monoid*, a set of finite sequences of zero or more elements from Σ. For example, for Σ=$\{a,b,c\}$, its Σ^* contains all concatenations of *a, b,* and *c*: $\{\lambda$*, a, ab, ba, cac, acbabc, ...*$\}$. The string of length zero is denoted by λ. w_i is the i^{th} word/string (w) of L. The length of a string u is denoted $|u|$. A stringset (or Formal Language) is a subset of Σ^*.

If u and v are strings, uv denotes their concatenation. For all $u,v,w,x \in \Sigma^*$, if x=uwv, then w is a *substring* of x. For example, *bc* is a *substring* of *abcd*, as concatenating *a,bc,d* yields *abcd*. Similarly, a string v is a *subsequence* of string w iff v=$\sigma_1\sigma_2...\sigma_n$ and $w \in \Sigma^*\sigma_1\Sigma^*\sigma_2\Sigma^*... \Sigma^*\sigma_n\Sigma^*$, where $\sigma \in \Sigma$. For example, string *bd* is a *subsequence* of length k=2 of *abcd*, *acd* is a subsequence

of length k=3 of the same string *abcd*, but string *db* is *not a subsequence* of *abcd*. A *subsequence* of length k is called a *k-subsequence*. Let $\text{subseq}_k(w)$ denote the set of subsequences of w up to length k.

A Strictly Piecewise grammar can be defined as a set of permissible subsequences. The grammar G is simply all strings whose k-long *subsequences* are permissible according to G. Consider a language L, consisting of Σ={*a,b,c,d*}. The grammar, G_{SP2}={*aa, ac, ad, ba, bb, bc, bd, ca, cb, cc, cd, da, db, dc, dd*} are comprised of these permissible *subsequences* of length k=2. Note, however, that although {*ab*} is a logically possible subsequence of length k, it is not in the grammar. Subsequences which are not in the grammar are called *forbidden strings*. The string u=[*bbcbdd*], where $|u|$=6 belongs to G_{SP2}, because it is composed of subsequences that are in that grammar. Similarly, the string v=[*bbdbbbcbddaa*], where $|v|$=12 belongs to G_{SP2}. However, the string w=[*bbabbbcbdd*] does not because {*ab*} is a forbidden subsequence as it is not part of the grammar. This condition applies for any string x for $|x| \in \mathbb{Z}$. One can also define an SP grammar for k=3 and k=4 for Σ={*a,b*} as $\text{G}_{\text{SP}3}$ and $\text{G}_{\text{SP}4}$ respectively. For example, $\text{G}_{\text{SP}3}$={*aaa, aab, abb, baa, bab, bba, bbb*}, with {*aba*} as *forbidden string*. A string [*aaaaaaab*] of length 8 is a valid G_{SP3} string and [*aaaaabaa*] is invalid. Thus, an appropriate grammar reflecting the dataset one intends to simulate can be designed by selecting appropriate permissible strings in the grammar. For the specific language, *forbidden strings* can be computed[1]. Note, to define an SPk grammar it is necessary to specify at least one *forbidden string*.

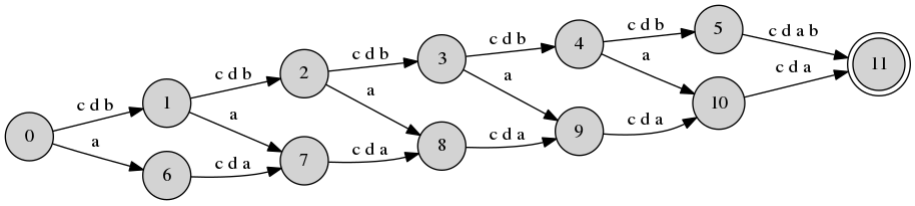


Fig. 1. The automaton for G_{SP2} (k=2) which generates strings of length=6

Figure 1 illustrates the finite-state diagram of a G_{SP2} for strings of length 6 with *forbidden string* {*ab*}. Traversing a path from state 1 until state 11 will generate valid G_{SP2} strings of length 6, e.g. {*accdda, caaaaa*}. It can also be noted that there is no path which generates a string which has an {*ab*} subsequence e.g. {*abcccc*} does not exist. Using the above described methodology, of choosing strings of appropriate length, one can simulate appropriate LDDs in a dataset. One can also control the number of dependent elements by choosing an appropriate k. *Forbidden strings* allow for elimination of certain combinations

[1] Refer section *5.2. Finding the shortest forbidden subsequences* in [16] for method to compute *forbidden sequences* for a particular SP language.

in generated datasets, which can be useful when one is trying to simulate real world datasets.

5 Experiment

In this experiment, we generate 4 datasets of SP*2* language. For each dataset we train an LSTM, an ORNN, and a RHN, and evaluate and compare the performance of the models.

5.1 Generating SP*2* dataset

For our experiment, Σ=$\{a,b,c,d\}$ was selected. *Forbidden strings* for this language were selected as $\{ab,bc\}$. In order to introduce various degrees of LDDs, strings with lengths l were generated in random order, where $2 \leq l \leq 500$. For every l, the number of strings per l is n_l. For this experiment, $n_l \leq$ 1,000,000. This allowed for uniform distribution of strings of all lengths. These strings were grouped in 4 datasets as described in Table 1. Within each dataset, strings were randomly ordered to avoid biased gradients. For training the neural networks, a subset of these generated datasets were used due to the size of each dataset.

Table 1. Datasets of SP*2* language

Dataset	Min Length	Max Length	Max LDDs	Original	Sample
Dataset 1	2	20	20	15 MB	15 MB
Dataset 2	21	100	100	470 MB	50 MB
Dataset 3	101	200	200	1.5 GB	100 MB
Dataset 4	201	500	500	9.9 GB	200 MB

The strings were generated using the tool *foma* [28]. A post processing *python* script was developed to select the small sample from the original datasets 1, 2, 3 and 4 as described in Table 1. Every dataset is made up of strings of varying l. The *python script* was also used to randomize the order of strings (as per the length), so as not to bias the models[2].

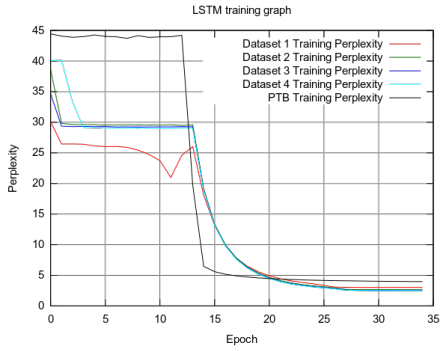
5.2 Training Task

All the networks were trained on a character prediction task. For each network type (LSTM, ORNN, RHN) a network was trained on each of the 4 SP*2* datasets, and also on a standard dataset of English language. The English language datasets were included in the experiments to provide a comparison for model performance when the vocabulary and type of data was varied. For the LSTM and ORNN the PTB was used as the standard English language dataset,

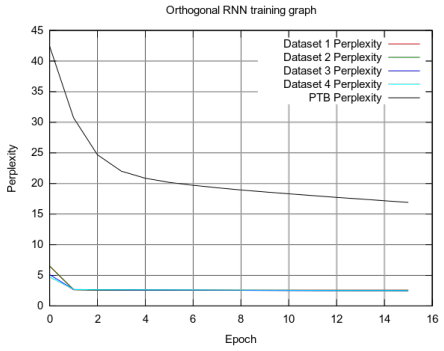
[2] Source code available at `https://github.com/silentknight/ICANN2018`

and for the RHN the Text8 dataset was used. Note, that the experimental task was kept constant across all datasets, so although the PTB and Text8 datasets are often used as part of a word-prediction task, in these experiments the networks were trained and evaluated on character prediction on the PTB and Text8 datasets. For SPk languages, the generated datasets were split into training (60%), validation (20%) and test (20%) sets. The LSTM[3] with dropout models were trained as advised in [29]; the ORNN[4] models were trained as recommended in [10]; and, the RHN[5] models were trained following [9].

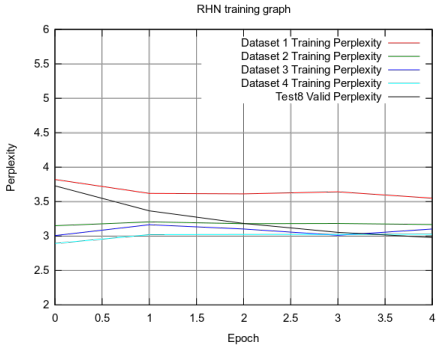
The performance of all the three network types was measured by computing the perplexity of the network after each epoch. The performance curve for the LSTM model is plotted in Figure 2.a., the performance of ORNN model is plotted in Figure 2.b., and the performance curve of RHN is plotted in Figure 2.c.



(a) LSTM Network



(b) Orthogonal RNN



(c) Recurrent Highway Network

Fig. 2. Perplexity vs training epoch for recurrent neural architectures.

[3] LSTM source `https://github.com/tensorflow/models/blob/master/tutorials/rnn/ptb/ptb_word_lm.py`

[4] ORNN Source `https://github.com/veugene/spectre_release`

[5] RHN source `https://github.com/julian121266/RecurrentHighwayNetworks`

6 Analysis

In Figure 2, we visualize the impact of increasing LDDs while training all the three architectures. Our results show that during the initial phase of the training, the LSTM network displayed perplexity directly proportional to the degree of LDDs present in the dataset. It is seen that dataset 4 (LDD order of around 500) presents higher perplexity as compared to the other datasets. However, every dataset eventually exhibits lower perplexity after epoch 20. When compared with the PTB task, one can observe lower perplexity by LSTM network in modeling datasets 1, 2, 3 and 4 during the initial phase of training. This is due in-part to the small vocabulary size in the SP*2* datasets (Σ=$\{a,b,c,d\}$). A small vocabulary size tends to lower entropy in a sequence. The PTB has much larger vocabulary thus increasing the entropy and eventually increasing perplexity. Selection of more *forbidden strings* leads to much richer grammar. SPk languages generated for this experiment contained only 2 *forbidden strings*, this led to generation of less rich grammar as compared to the PTB (English grammar). However, one can observe that the LSTM model learns the PTB much faster than SP*2* languages (the graph drops earlier). This can be directly attributed to the presence of longer LDDs in the SP*2* datasets.

Orthogonal RNNs enforce soft orthogonality to address vanishing gradient problem. When compared with LSTM network training of the PTB, it is observed that the perplexity of both architectures is very similar during the initial training phase, but ORNNs performance does not improve with more training as compared to LSTM. The impact of vocabulary size is also evident in this case (the perplexity for PTB is much higher than for the SP*2* datasets). However, it can be seen that ORNNs trained with datasets 1 and 2 present higher perplexity as compared to datasets 3 and 4 (longer LDDs) suggesting that ORNN models overfit datasets 1 and 2 and are able to generalize to datasets 3 and 4. This could be attributed to orthogonal weight initializations which makes learning longer dependencies easier.

Focusing on the graph for the Recurrent Highway Networks it can be observed that the model tended to exhibit lower perplexity on SP*2* datasets with higher degrees of LDDs. This could be attributed to the architecture of the network. Due to increased depth in recurrent transitions in these networks, it was possible for the model to achieve good performance on datasets with long LDDs. However, on datasets with lower degrees of LDDs these models tend to overfit and, thus, exhibit higher perplexity. Furthermore, comparing the RHN graph on the Text8 dataset with the LSTM and ORNN graphs on the PTB it is apparent that RHNs are better at handling larger vocabularies: the RHN graph for Text8 is lower than the LSTM and ORNN graphs on the PTB.

7 Conclusion

In this paper, we used SPk languages to generate benchmarking datasets for LDDs. We trained various RNNs with the generated datasets and analyzed their

performance. The analysis revealed that SPk languages are able to generate datasets with varying degree of LDDs. Consequently, using SPk languages gives experimental control over the generation of rich datasets by controlling the k, the length of the strings, the vocabulary of the generated language, and by choosing appropriate *forbidden strings*. The analysis also revealed that RHNs have a much better capability (as compared with LSTMs and ORNNs) to model LDDs.

Acknowledgements This research was partly supported by the ADAPT Research Centre, funded under the SFI Research Centres Programme (Grant 13/RC/2106) and is co-funded under the European Regional Development Funds. The research was also supported by an IBM Shared University Research Award. We also, gratefully, acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU under NVIDIA GPU Grant used for this research.

References

1. Elman, J. L.: Finding Structure in Time. Cognitive Science **14**, pp. 179-211 (1990).
2. Hochreiter. S.: Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, TU Munich (1991).
3. Yoshua, B., Simard, P., Frasconi, P.: Learning Long-Term Dependencies with Gradient Descent is Difficult. IEEE Transactions on Neural Networks **5**(2), pp. 157-166 (1994).
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation, **9**(8), pp. 1735-1780, (1997).
5. Graves, A., Wayne, G., Danihelka, I.: Neural Turing Machines. CoRR (2014).
6. Salton, G. D., Ross, R. J., Kelleher, J. D.: Attentive Language Models. In: Proceedings of the The 8th International Joint Conference on Natural Language Processing, pp. 441-450 (2017).
7. Merity, S., Xiong, C., Bradbury, J., Socher, R.: Pointer Sentinel Mixture Models. In: ICLR 2016, (2016).
8. Chang, S. et al.: Dilated Recurrent Neural Networks. In: Editors Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.), Advances in Neural Information Processing Systems 30, pp. 77-87. Curran Associates, Inc. (2017).
9. Zilly, J. G., Srivastava, R. K., Koutnk, J., Schmidhuber, J.: Recurrent Highway Networks. In: Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR **70** (2017).
10. Vorontsov, E., Trabelsi, C., Kadoury, S., Pal, C.: On orthogonality and learning recurrent networks with long term dependencies. In: Proceeding of ICML 2017 (2017).
11. Henaff, M., Szlam, A., LeCun, Y.: Recurrent Orthogonal Networks and Long-Memory Tasks. In: Proceedings of The 33rd International Conference on Machine Learning, in PMLR **48**, pp. 2034-2042 (2016).
12. Marcus, M. P., Marcinkiewicz, M. A., Santorini, B.: Building a large annotated corpus of english: The penn treebank. Computational Linguistics **19**(2), pp. 313-330 (1993). ISSN 0891-2017.
13. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE **86**(11), pp. 2278-2324 (1998).

14. Rogers, J., Heinz, J., Bailey, G., Edlefsen, M., Visscher, M., Wellcome, D., Wibel, S.: On Languages Piecewise Testable in the Strict Sense. In: Ebert C., Jger G., Michaelis J. (eds.) The Mathematics of Language. Lecture Notes in Computer Science, **6149**, Springer, Berlin, Heidelberg (2010).
15. Simon, I.: Piecewise testable events. In: Automata Theory and Formal Languages, pp. 214-222 (1975).
16. Fu, J., Heinz, J., Tanner, H.G.: An Algebraic Characterization of Strictly Piecewise Languages. In: Ogihara M., Tarui J. (eds.) Theory and Applications of Models of Computation. TAMC 2011. Lecture Notes in Computer Science, vol. 6648, Springer, Berlin, Heidelberg (2011).
17. Avcu, E., Shibata, C., Heinz, J.: Subregular Complexity and Deep Learning. In: Proceedings of the Conference on Logic and Machine Learning in Natural Language (LaML 2017), **1**, pp. 20-33 (2017).
18. Heinz. J., Rogers, J.: Estimating Strictly Piecewise Distributions. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pp. 886-896 (2010).
19. Reber, A. S.: Implicit learning of artificial grammars. Journal of Verbal Learning and Verbal Behavior **6**(6), pp.855-863 (1967).
20. Tomita, M.: Learning of construction of finite automata from examples using hill-climbing. In: Proceedings of Fourth International Cognitive Science Conference, pp. 105-108 (1982).
21. Casey, M.: The dynamics of discrete-time computation, with application to recurrent neural networks and finite statemachine extraction. Neural computation **8**(6), pp. 1135-78 (1996).
22. Smith, A. W., Zipser, D.: Encoding sequential structure: experience with the real-time recurrent learning algorithm. In: Proceedings of IJCNN, vol. I, pp. 645-648 (1989).
23. Chomsky, N.: Three models for the description of language. IRE Transactions on Information Theory **2**, pp 113-124 (1956).
24. Chomsky, N.: On certain formal properties of grammars. Information Control **2**, pp 137-167 (1959).
25. Fitch, W. T., Friederici, A. D.: Artificial grammar learning meets formal language theory: an overview. Philosophical Transactions of Royal Society B: Biological Sciences **367**(1598), pp. 1933-1955 (2012).
26. Jager, G., Rogers, J.: Formal language theory: refining the Chomsky hierarchy. Philosophical Transactions of Royal Society B: Biological Sciences **367**(1598), pp. 1956-1970 (2012).
27. Ebeling, W., Poeschel, T.: Entropy and Long range correlations in literary English. Europhysics Letters **26**(2), pp. 241-246 (1994).
28. Hulden, M.: Foma: a finite-state compiler and library. Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, pp. 29-32 (2009).
29. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent Neural Network Regularization. In: Proceedings of ICRL (2015).