Can Neural Networks Learn Symbolic Rewriting?

Bartosz Piotrowski [1 2] **Josef Urban** [2] **Chad E. Brown** [2] **Cezary Kaliszyk** [3 1]

Abstract

This work investigates if the current neural architectures are adequate for learning symbolic rewriting. Two kinds of data sets are proposed for this research – one based on automated proofs and the other being a synthetic set of polynomial terms. The experiments with use of the current neural machine translation models are performed and its results are discussed. Ideas for extending this line of research are proposed and its relevance is motivated.

1. Introduction

Neural networks (NNs) turned out to be very useful in several domains. In particular, one of the most spectacular advances achieved with use of NNs has been natural language processing. One of the tasks in this domain is translation between natural languages – neural machine translation (NMT) systems established here the state-of-the-art performance. Recently, NMT produced first encouraging results in the *autoformalization task* (Kaliszyk et al., 2014; 2015; 2017; Wang et al., 2018) where given an *informal* mathematical text in LaTeX the goal is to translate it to its *formal* (computer understandable) counterpart. In particular, the NMT performance on a large synthetic LaTeX-to-Mizar dataset produced by a relatively sophisticated toolchain developed for several decades (Bancerek et al., 2018) is surprisingly good (Wang et al., 2018), indicating that neural networks can learn quite complicated algorithms for symbolic data. This inspired us to pose a question: *Can NMT models be used in the formal-to-formal setting?* In particular: *Can NMT models learn symbolic rewriting?*

The answer is relevant to various tasks in automated reasoning. For example, neural models could compete with symbolic methods such as inductive logic programming (Muggleton & De Raedt, 1994) (ILP) that have been previously experimented with to learn simple rewrite tasks and theorem-proving heuristics from large formal corpora (Urban, 1998). Unlike (early) ILP, neural methods can however easily cope with large and rich datasets, without combinatorial explosion.

Our work is also an inquiry into the capabilities of NNs as such, in the spirit of works like (Evans et al., 2018).

2. Data

To perform experiments answering our question we prepared two data sets – the first consists of examples extracted from proofs found by ATP (automated theorem prover) in a mathematical domain (AIM loops), whereas the second is a synthetic set of polynomial terms.

2.1. The AIM data set

The data consists of sets of ground and nonground rewrites that came from `Prover9`[1] proofs of theorems about AIM loops produced by Veroff (Kinyon et al., 2013).

Many of the inferences in the proofs are paramodulations from an equation and have the form

$$\frac{s = t \qquad u[\theta(s)] = v}{u[\theta(t)] = v}$$

where s, t, u, v are terms and θ is a substitution. For the most common equations $s = t$, we gathered corresponding pairs of terms $\big(u[\theta(s)], u[\theta(t)]\big)$ which were rewritten from one to another with $s = t$. We put the pairs to separate data sets (depending on the corresponding $s = t$): in total 8 data sets for ground rewrites (where θ is trivial) and 12 for nonground ones. The goal will be to learn rewriting for each of this 20 rules separately.

Terms in the examples are treated as linear sequences of tokens where tokens are single symbols (variable / costant / predicate names, brackets, commas). Numbers of examples in each of the data sets vary between 251 and 34101. Lengths of the sequences of tokens vary between 1 and 343, with mean around 35. These 20 data sets were split into training, validation and test sets for our experiments ($60\%, 10\%, 30\%$, respectively).

[*]Equal contribution [1]University of Warsaw, Poland [2]Czech Technical University, Prague [3]University of Innsbruck, Austria. Correspondence to: Bartosz Piotrowski <bartoszpiotrowski@post.pl>, Josef Urban <josef.urban@gmail.com>.

Presented at the ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data Copyright 2019 by the author(s).

[1]https://www.cs.unm.edu/~mccune/prover9/

Table 1. Example of a ground rewrite in the AIM data set.

Rewrite rule:	`b(s(e, v1), e) = v1`
Before rewriting:	`k(b(s(e, v1), e), v0)`
After rewriting:	`k(v1, v0)`

Table 2. Example of a nonground rewrite in the AIM data set.

Rewrite rule:	`o(V0, e) = V0`
Before rewriting:	`t(v0, o(v1, o(v2, e)))`
After rewriting:	`t(v0, o(v1, v2))`

In Table 1 and Table 2 there are presented examples of pairs of AIM terms in TPTP (Sutcliffe, 2017) format, before and after rewriting with, respectively, ground and nonground rewrite rules.[2]

2.2. The polynomial data set

This is a synthetically created data set where the examples are pairs of equivalent polynomial terms. The first element of each pair is a polynomial in an arbitrary form and the second element is the same polynomial in a normalized form. The arbitrary polynomials are created randomly in a recursive manner from a set of available (non-nullary) function symbols, variables and constants. First, one of the symbols is randomly chosen. If it is a constant or a variable it is returned and the process terminates. If a function symbol is chosen, its subterm(s) are constructed recursively in a similar way.

The parameters of this process are set in such a way that it creates polynomial terms of average length around 25 symbols. Terms longer than 50 are filtered out. Several data sets of various difficulty were created by varying the number of available symbols. This were quite limited – at most 5 different variables and constants being a few first natural numbers. The reason for this limited complexity of the input terms is because normalizing even a relatively simple polynomial can result in a very long term with very large constants – which is related especially to the operation of exponentiation in polynomials.

Each data set consists of different 300 000 examples, see Table 3 for examples. These data sets were split into training, validation and test sets for our experiments (60%, 10%, 30%, respectively).[3]

[2]All the described AIM data are available at https://github.com/BartoszPiotrowski/rewriting-with-NNs/tree/master/data/AIM

Table 3. Examples in the polynomial data set.

Before rewriting:	After rewriting:
`(x*(x+1))+1`	`x^2+x+1`
`(2*y)+(1+(y*y))`	`y^2+2*y+1`
`(x+2)*(((2*x)+1)+(y+1))`	`2*x^2+5*x+y+3`

3. Experiments

For experiments with both data sets we used an established NMT architecture (Luong et al., 2017) based on LSTMs (long short-term memory cells) and implementing the attention mechanism.[4]

After a small grid search we decided to inherit most of the hyperparameters of the model from the best results achieved in (Wang et al., 2018) where LaTeX-to-Mizar translation is learned. We used relatively small LSTM cells consisting of 2 layers with 128 units. The "scaled Luong" version of the attention mechanism was used, as well as dropout with rate equal 0.2. The number of training steps was 10000. (This setting was used for all our experiments described below.)

3.1. AIM data set

First, NMT models were trained for each of the 20 rewrite rules in the AIM data set. It turned out that the models, as long as the number of examples was greater than 1000, were able to learn the rewriting task very well, reaching 90% of accuracy on separated test sets. This means that the task of applying single rewrite step seems relatively easy to learn by NMT. See Table 4 for all the results.

We also run an experiment on the joint set of all rewrite rules (consisting of 41396 examples). Here the task was more difficult as a model needed not only to apply rewriting correctly, but also choose "the right" rewrite rule applicable for a given term. Nevertheless, the performance was also very good, reaching 83% of accuracy.

3.2. Polynomial data set

Then experiments on more challenging but also much larger data sets for polynomial normalization were performed. Depending on the difficulty of the data, accuracy on the test sets achieved in our experiments varied between 70% and 99%. The results in terms of accuracy are shown in Table 5.

This high performance of the model encouraged a closer inspection of the results. First, we checked if in the test sets there are input examples which differs from these in training sets only by renaming of variables. Indeed, for each

[3]The described polynomial data are available at https://github.com/BartoszPiotrowski/rewriting-with-NNs/tree/master/data/polynomial

[4]We also experimented with the Transformer model (Vaswani et al., 2017) but the results were worse. This could be due to a limited grid search we performed as Transformer is known to be very sensitive to hyperparameters.

Table 4. Results of experiments with AIM data. (Names of the rules correspond to folder names in the Github repo.)

Rule:	Training examples:	Test examples:	Accuracy on test:
`abstrused1u`	2472	1096	86.5%
`abstrused2u`	2056	960	89.2%
`abstrused3u`	1409	666	84.3%
`abstrused4u`	1633	743	87.4%
`abstrused5u`	2561	1190	89.5%
`abstrused6u`	81	40	12.5%
`abstrused7u`	76	37	0.0%
`abstrused8u`	79	39	2.5%
`abstrused9u`	1724	817	86.7%
`abstrused10u`	3353	1573	82.9%
`abstrused11u`	10230	4604	79.0%
`abstrused12u`	7201	3153	87.2%
`instused1u`	198	97	20.6%
`instused2u`	196	87	25.2%
`instused3u`	83	41	29.2%
`instused4u`	105	47	2.1%
`instused5u`	444	188	59.5%
`instused6u`	1160	531	87.5%
`instused7u`	307	144	13.8%
`instused8u`	116	54	3.7%
union of all	41396	11826	83.2%

of the data sets in test sets are $5 - 15\%$ of such "renamed" examples. After filtering them out the measured accuracy drops – but only by $1 - 2\%$.

An examination of the examples wrongly rewritten by the model was done. It turns out that the wrong outputs almost always parse (in $97-99\%$ of cases they are legal polynomial terms). Notably, depending on the difficulty of the data set, as much as $18 - 64\%$ of incorrect outputs are wrong only with respect to the constants in the terms. (Typically, NMT model proposes too low constants compared to the correct ones.) Below 1% of wrong outputs are correct modulo variable renaming.

Table 5. Choosen results of experiments with polynomials. (Characteristic of formulas concerns the *input* polynomials. Labels of the data sets correspond to folder names in the Github repo.)

Label	Function symbols	Constant symbols	Num. of variables	Accuracy on test
`poly1`	$+, *$	$0, 1$	1	99.2%
`poly2`	$+, *$	$0, 1$	2	97.4%
`poly3`	$+, *$	$0, 1$	3	88.2%
`poly4`	$+, *$	$0, 1, 2, 3, 4, 5$	5	83.4%
`poly5`	$+, *, \hat{}$	$0, 1$	2	85.5%
`poly6`	$+, *, \hat{}$	$0, 1, 2$	3	71.8%

4. Conclusions and future work

NMT is not typically applied to symbolic problems, but surprisingly, it performed very well for both described tasks. The first one was easier in terms of complexity of the rewriting (only one application of a rewrite rule was performed) but the number of examples was quite limited. The second task involved more difficult rewriting – multiple different rewrite steps were performed to construct the examples. Nevertheless, provided many examples, NMT could learn normalizing polynomials.

We hope this work provides a baseline and inspiration for continuing this line of research. We see several interesting directions this work can be extended.

Firstly, more interesting and difficult rewriting problems need to be provided for better delineation of the strength of the neural models. The described data are relatively simple and with no direct relevance to the real unsolved symbolic problems. But the results on these simple problems are encouraging enough to try with more challenging ones, related to real difficulties – e.g. these from TPDB data base.[5]

Secondly, we are going to develop and test new kinds of neural models tailored for the problem of comprehending symbolic expressions. Specifically, we are going to implement an approach based on the idea of TreeNN, which may be another effective approach for this kind of tasks (Evans et al., 2018; Miao et al., 2018; Chakraborty et al., 2018). TreeNNs are built recursively from *modules*, where the modules corresponds to parts of symbolic expression (symbols) and the shape of the network reflects the parse tree of the processed expression. This way model is explicitly informed on the exact structure of the expression, which in case of formal logic is always unambiguous and easy to extract. Perhaps this way the model could learn more efficiently from examples (and achieve higher results even on the small AIM data sets). The authors have a positive experience of applying TreeNNs to learn remainders of arithmetical expressions modulo small natural numbers – TreeNNs outperformed here neural models based on LSTM cells, giving almost perfect accuracy. However, this is unclear how to translate this TreeNN methodology to the tasks with the structured output, like the symbolic rewriting task.

Thirdly, there is an idea of integrating neural rewriting architectures into the larger systems for automated reasoning. This can be motivated by the interesting contrast between some simpler ILP systems suffering for combinatorial explosion in presence of a large number of examples and neural methods which definitely benefit form large data sets.

We hope that this work will inspire and trigger a discussion on the above (and other) ideas.

[5] `http://termination-portal.org/wiki/TPDB`

Acknowledgements

Piotrowski was supported by the grant of National Science Center, Poland, no. 2018/29/N/ST6/02903, and by the European Agency COST action CA15123. Urban and Brown were supported by the ERC Consolidator grant no. 649043 *AI4REASON* and by the Czech project *AI&Reasoning* CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund. Kaliszyk was supported by ERC Starting grant no. 714034 *SMART*.

References

Bancerek, G., Naumowicz, A., and Urban, J. System description: XSL-based translator of Mizar to LaTeX. In Rabe, F., Farmer, W. M., Passmore, G. O., and Youssef, A. (eds.), *11th International Conference on Intelligent Computer Mathematics (CICM 2018)*, volume 11006 of *LNCS*, pp. 1–6. Springer, 2018. URL https://doi.org/10.1007/978-3-319-96812-4_1.

Chakraborty, S., Allamanis, M., and Ray, B. Tree2tree neural translation model for learning source code changes. *CoRR*, abs/1810.00314, 2018. URL http://arxiv.org/abs/1810.00314.

Evans, R., Saxton, D., Amos, D., Kohli, P., and Grefenstette, E. Can neural networks understand logical entailment? In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SkZxCk-0Z.

Kaliszyk, C., Urban, J., Vyskočil, J., and Geuvers, H. Developing corpus-based translation methods between informal and formal mathematics: Project description. In Watt, S. M., Davenport, J. H., Sexton, A. P., Sojka, P., and Urban, J. (eds.), *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, volume 8543 of *LNCS*, pp. 435–439. Springer, 2014. URL http://dx.doi.org/10.1007/978-3-319-08434-3_34.

Kaliszyk, C., Urban, J., and Vyskočil, J. Learning to parse on aligned corpora (rough diamond). In Urban, C. and Zhang, X. (eds.), *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, volume 9236 of *LNCS*, pp. 227–233. Springer, 2015. URL http://dx.doi.org/10.1007/978-3-319-22102-1_15.

Kaliszyk, C., Urban, J., and Vyskocil, J. Automating formalization by statistical and semantic parsing of mathematics. In Ayala-Rincón, M. and Muñoz, C. A. (eds.), *8th International Conference on Interactive Theorem Proving (ITP 2017), Brasília, Brazil, September 26-29, 2017, Proceedings*, volume 10499 of *LNCS*, pp. 12–27. Springer, 2017. URL https://doi.org/10.1007/978-3-319-66107-0_2.

Kinyon, M. K., Veroff, R., and Vojtechovský, P. Loops with abelian inner mapping groups: An application of automated deduction. In Bonacina, M. P. and Stickel, M. E. (eds.), *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *LNCS*, pp. 151–164. Springer, 2013. URL https://doi.org/10.1007/978-3-642-36675-8_8.

Luong, M., Brevdo, E., and Zhao, R. Neural machine translation (seq2seq) tutorial, 2017. URL https://github.com/tensorflow/nmt.

Miao, N., Wang, H., Le, R., Tao, C., Shang, M., Yan, R., and Zhao, D. Tree2tree learning with memory unit, 2018. URL https://openreview.net/forum?id=Syt0r4bRZ.

Muggleton, S. and De Raedt, L. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

Sutcliffe, G. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

Urban, J. Experimenting with machine learning in automatic theorem proving. Master's thesis, Faculty of Mathematics and Physics, Charles University, Prague, 1998. English summary at https://www.ciirc.cvut.cz/~urbanjo3/MScThesisPaper.pdf.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 6000–6010, 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.

Wang, Q., Kaliszyk, C., and Urban, J. First experiments with neural translation of informal to formal mathematics. In Rabe, F., Farmer, W. M., Passmore, G. O., and Youssef, A. (eds.), *11th International Conference on Intelligent Computer Mathematics (CICM 2018)*, volume 11006 of *LNCS*, pp. 255–270. Springer, 2018. URL https://doi.org/10.1007/978-3-319-96812-4_22.