Can Neural Networks Learn Symbolic Rewriting?

Bartosz Piotrowski, Chad Brown, Josef Urban
and Cezary Kaliszyk

7 April 2019, Obergurgl

Motivation

- Neural network architectures proved to be very successful in various tasks related to processing natural language; more notably, *neural machine translation* systems (NMT) established state-of-the-art in the task of translation between languages.
- Recently, NMT produced first encouraging results in the autoformalization task where given an informal mathematical text in LaTeX the goal is to translate it to its formal (computer understandable) counterpart.
    - See *Wang, Kaliszyk, Urban. First experiments with neural translation of informal to formal mathematics. CICM 2018.*
- This encouraged us to pose a question:

  Can NMT models learn symbolic rewriting?

Can NMT models learn symbolic rewriting?

An attempt to answer this question is important for several reasons:

1. It may allow for better understanding of the capabilities and limitations of the current neural network architectures.
   - This is in the spirit of works like *Evans et al., Can Neural Networks Understand Logical Entailment? ICLR 2018.*
2. It may be relevant for various tasks in automated reasoning. Neural models could compete with symbolic methods such as *inductive logic programming* (ILP) that have been previously experimented with to learn simple rewrite tasks and theorem-proving heuristics from large formal corpora.
   - There is a striking contrast between ILP and NMT methods with respect to handling large and rich data sets: ILP can suffer for combinatorial explosion whereas for NMT much data is beneficial.
3. It can motivate developing new kinds of neural architectures.

Data sets

To perform experiments answering our question we prepared two different data sets:

1. A set of examples found in ATP proofs in a mathematical domain – AIM loops (*Abelian inner mappings*).
2. A synthetic set of polynomial terms.

AIM data set

- The data consists of sets of ground and non-ground rewrites that came from `Prover9` proofs of theorems about AIM loops produced by Bob Veroff and Michael Kinyon.
- 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 non-ground ones.

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)`

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)`

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)`

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))`

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))`

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))`

AIM data set

Rewrite rule:	`k(V0, k(V1, V2)) = k(V1, k(V0, V2))`
Before rewriting:	`l(k(v1, k(v0, v2)), k(v0, v2), v3)`
After rewriting:	`l(k(v0, k(v1, v2)), k(v0, v2), v3)`

AIM data set

Rewrite rule:	`k(V0, k(V1, V2)) = k(V1, k(V0, V2))`
Before rewriting:	`l(k(v1, k(v0, v2)), k(v0, v2), v3)`
After rewriting:	`l(k(v0, k(v1, v2)), k(v0, v2), v3)`

AIM data set

Rewrite rule:	`k(V0, k(V1, V2)) = k(V1, k(V0, V2))`
Before rewriting:	`l(k(v1, k(v0, v2)), k(v0, v2), v3)`
After rewriting:	`l(k(v0, k(v1, v2)), k(v0, v2), v3)`

AIM data set

- Each of the 20 rewrite rules corresponds to a data set with number of examples (pairs of terms) between 150 and 11000.
- We also took a union of all these data sets which gave $\sim$ 53000 examples.
- The data sets were split into training (70%) and test (30%) sets.

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 the normalized form.
- The arbitrary polynomials are created randomly in a recursive manner from a set of available (non-nullary) function symbols, variables and constants.

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`

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`

Polynomial data set

- Several data sets of various difficulty were created by varying
  1. the number of available symbols,
  2. the length of the polynomials.
- Each created data set consists of 300000 examples.
- The data sets were split into training (70%) and test (30%) sets.

Experiments

- For experiments we used an established NMT implementation from the Tensorflow repo (`https://github.com/tensorflow/nmt`).
- This NMT implementation is a classical sequence-to-sequence architecture based on LSTM cells and using the attention mechanism.
- Hyperparameters used for training were inherited from experiments on LaTeX-to-Mizar translation by Shawn et al.
- (Additionally, we experimented with the Transformer model which is a sequence-to-sequence model not using recurrent connections but only multi-head attention (see *Vaswani et al,. Attention Is All You Need. NIPS 2017*). After training for the same number of steps the achieved results were weaker. But we didn't tune paramteres too much and Transformer is very sensitive to hyperparameters.)

Experiments

Some of the hyperparameters of NMT which were used:

```
--num_train_steps=10000
--attention=scaled_luong
--num_layers=2
--num_units=128
--dropout=0.2
```

Results for AIM data set

- We trained NMT models for each of the 20 rewrite rules in the AIM data set.
- Additionally, we trained an NMT model on a joint set of all rewrite rules.
- As long as the number of examples was greater than 1000, were able to learn the rewriting task with high accuracy – reaching $\sim$ 90% on separated test sets.
- On the joint set of all rewrite rules (consisting of 41396 examples) the performance was also good – 83%.
- This means that the task of applying single rewrite step seems relatively easy to learn by NMT.

Results for AIM data set

Rule:	Training examples:	Test examples:	Accuracy:
abstrused1u	2472	1096	86.50%
abstrused2u	2056	960	89.27%
abstrused3u	1409	666	84.38%
abstrused4u	1633	743	87.48%
abstrused5u	2561	1190	89.58%
abstrused6u	81	40	12.50%
abstrused7u	76	37	0.00%
abstrused8u	79	39	2.56%
abstrused9u	1724	817	86.78%
abstrused10u	3353	1573	82.96%
abstrused11u	10230	4604	79.00%
abstrused12u	7201	3153	87.22%
instused1u	198	97	20.62%
instused2u	196	87	25.29%
instused3u	83	41	29.27%
instused4u	105	47	2.13%
instused5u	444	188	59.57%
instused6u	1160	531	87.57%
instused7u	307	144	13.89%
instused8u	116	54	3.70%
union of all	41396	11826	83.29%

Results for AIM data set

Rule:	Training examples:	Test examples:	Accuracy:
abstrused1u	**2472**	1096	**86.50%**
abstrused2u	**2056**	960	**89.27%**
abstrused3u	**1409**	666	**84.38%**
abstrused4u	**1633**	743	**87.48%**
abstrused5u	**2561**	1190	**89.58%**
abstrused6u	81	40	12.50%
abstrused7u	76	37	0.00%
abstrused8u	79	39	2.56%
abstrused9u	**1724**	817	**86.78%**
abstrused10u	**3353**	1573	**82.96%**
abstrused11u	**10230**	4604	**79.00%**
abstrused12u	**7201**	3153	**87.22%**
instused1u	198	97	20.62%
instused2u	196	87	25.29%
instused3u	83	41	29.27%
instused4u	105	47	2.13%
instused5u	444	188	59.57%
instused6u	**1160**	531	**87.57%**
instused7u	307	144	13.89%
instused8u	116	54	3.70%
union of all	**41396**	11826	**83.29%**

Results for polynomial data set

- The polynomial data set appeared to be more challenging but also was much larger.
- The results were rather very satisfying – depending on the difficulty of the data, accuracy on the test sets achieved in our experiments varied between 70% and 99%.

Results for polynomial data set

Function symbols	Constant symbols	Number of variables	Maximum length	Accuracy on test
$+, *$	$0, 1$	1	30	99.28%
$+, *$	$0, 1$	2	30	97.43%
$+, *$	$0, 1$	3	50	88.20%
$+, *$	$0, 1, 2, 3, 4, 5$	5	50	83.47%
$+, *,$ ^	$0, 1$	2	50	85.56%
$+, *,$ ^	$0, 1, 2$	3	50	71.81%

Conclusions

- NMT is not typically applied to symbolic problems, but somewhat 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.

Future work

We propose several directions in which this work can be extended:

- Experimenting with more interesting and challenging problems related to rewriting.
- Implementing new neural architectures suited especially for this kind of symbolic problems. In particular, we want to implement architectures whose structure is conditioned on a tree shape of the terms. (*TreeNN-based* models and their extensions, e.g. with attention mechanism.)
- Find some task where it would be interesting to compare performance of NMT-based rewriting with ILP-based rewriting.
- Find a way how to fruitfully use NMT methods within automated reasoning systems.

Thank you!

Can Neural Networks Learn Symbolic Rewriting?

It seems that, in some sense, yes!

Bartosz Piotrowski, Chad Brown, Josef Urban
and Cezary Kaliszyk

7 April 2019, Obergurgl