Deep Learning for Automated Theorem Proving

Stanisław J. Purgał[1,*,†]

[1]University of Innsbruck, Innsbruck, Tirol, Austria

Abstract

The field of Artificial Intelligence has seen great advances with the use of Deep Neural Networks. However, the problem of creating a system capable of abstract reasoning remains unsolved. One way to force AI to perform abstract reasoning is to directly learn an abstract task – such as Automated Theorem Proving. In ATP, the goal is to construct a formal proof of a mathematical statement. This requires finding a correct sequence of inferences in an exponentially large space of possibilities. This search can be guided by a Deep Neural Network. Towards this end, I have worked on creating a neural architecture that would work well with mathematical formulas and providing a way to generate training data to train such neural networks.

1. Introduction

In some ways, machine learning is a natural fit for theorem proving. For one, we do not really need to care about testing robustness [1], as we have a built-in, absolute way of testing performance - are the correct proofs successfully constructed. While it may be useful during research, we do not really need to understand why do our heuristics give the answers they do, as long as we can verify the final proof. There are no consequences for the predictions being inaccurate, other than the failure to construct a proof.

Another reason to turn to machine learning is the fact that the problems in theorem proving are in general undecidable. If we still want to be able to solve such problems automatically, we have to use heuristics. Hand crafted methods rarely (if ever) reach the performance of a human working on a problem. At the same time, some machine learning methods achieved and surpassed human-level performance for some tasks [2].

2. Research questions

In applying deep neural networks to theorem proving I see two main challenges:

1. providing a neural network architecture that works well with formal mathematical descriptions
2. providing a way to train such network, if possible without relying on human input

CICM 2021 Doctoral Program

stanislaw.purgal@uibk.ac.at (S. J. Purgał)

3. Neural models for formulas

One natural idea for processing mathematical concepts with neural networks is to represent them as graphs. However, unlike image processing, where convolutions and pooling is a well performing standard, there is no such architecture for Graph Neural Networks (GNN).

Commonly used pattern is *message passing*, where every layer combines information from neighbouring nodes to compute next layer's node embeddings. This method however has several drawbacks. For one, information can only travel one edge per layer, so combining information from far away nodes requires many layers. This problem is made worse by the fact that GNNs tend to perform badly with large number of layers. Another problem is the fact that such network can recognize graphs only up to Weisfeiler–Lehman isomorphism test [3, 4], meaning that if the test says the graphs are isomorphic, the networks will process the graphs as if they really were exactly the same — even if they are not.

My work [5] seeks to overcome these problems by adding two modification to standard message-passing scheme.

- expanding window – in every layer this network aggregates information from nodes exponentially far away (in ith layer use nodes within 2^i distance)
- random identifiers – as part initial (0th layer) embedding nodes are given a random sequence of 1s and 0s. This allows the network to tell the difference between being connected to the same node and being connected to a node with the same neighbourhood. A similar idea was also explored in [6].

While this approach does solve the aforementioned problems, it does not seem to produce any significant improvement in final performance.

I also developed a formula embedding [7], that that would allow extracting subformulae with linear transformations (thus easily done by neural layers). We do it by training an autoencoder. We test these embeddings by training a simple feed-forward network to perform premise selection classification, using learned earlier embeddings. This achieves 70% accuracy, while SotA on this dataset is 81% (when training directly on the dataset).

4. Self-play

To actually make use of a neural network we need to be able to provide it with data. One obvious approach is to use human-generated data and do imitation learning. However, if we want to be able to solve difficult problems, merely imitating humans might not be enough.

It would be better to learn from the ground truth, that is straight from mathematical results. Some attempts at this have been made in the meantime, like [8, 9] where an algorithm tries to prove theorems from a given dataset and learns from successful attempts – this however requires a robust dataset of theorems and presents a clear limit of what can be learned. A different attempt [10] generates a dataset of synthetic theorems and learns from it, thus removing the need for a human provided dataset, however this allows for only a shallow exploration of possible theorems.

My first attempt at this problem uses an algorithm that was already proven to effectively explore and learn an extensive field – the board game of Go. The AlphaZero [2] algorithm learns

by playing the game against itself, then uses the generated games to learn and improve. Then it generates better games and so on. This allows a deep exploration towards interesting and valuable areas in the space of possible games by continuously using already learned knowledge to learn more.

In my work, I train a neural network to play a game, where one player generates a provable theorem and the other proves it. Here, improvement of one driven improvement of the other in a continuous learning process.

The main theoretical problem is the goal of the training, which is finding theorems that are provable but hard to prove. This should result in the adversary (the player constructing theorems) learning some way to construct uninteresting but hard theorems, and then forever winning the game. This would cause the prover to forget everything it learned in a vain attempt to learn to prove theorems which it can never learn to prove because of computational limits.

The interesting question then becomes, how smart can a prover get before this happens? In my experiments however, this does not seem to happen, as the neither the prover not adversary seem to get smart enough, instead producing simple but obfuscated theorems. As of yet I cannot answer whether it is due to my lackluster implementation, small computational resources (compared to AlphaZero experiments) or some problem inherent to the algorithm.

5. Future research

Beyond the two research results discussed above [5, 7], my future research involves working on the theoretical problem mentioned above, to somehow change the objective from finding hard to prove theorems. One such objective would be finding theorems useful in proving other theorems.

References

[1] S. Gu, L. Rigazio, Towards deep neural network architectures robust to adversarial examples, 2014. arXiv:1412.5068.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, nature 529 (2016) 484.

[3] B. Weisfeiler, A. A. Lehman, A reduction of a graph to a canonical form and an algebra arising during this reduction, Nauchno-Technicheskaya Informatsia 2 (1968) 12–16.

[4] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, arXiv preprint arXiv:1810.00826 (2018).

[5] S. Purgał, Improving expressivity of graph neural networks, CoRR abs/2004.05994 (2020). URL: https://arxiv.org/abs/2004.05994. arXiv:2004.05994.

[6] G. Dasoulas, L. D. Santos, K. Scaman, A. Virmaux, Coloring graph neural networks for node disambiguation, CoRR abs/1912.06058 (2019). URL: http://arxiv.org/abs/1912.06058. arXiv:1912.06058.

[7] S. Purgał, J. Parsert, C. Kaliszyk, A study of continuous vector representations for theorem proving, Journal of Logic and Computation (2021).

URL: https://doi.org/10.1093/logcom/exab006. doi:10.1093/logcom/exab006. arXiv:https://academic.oup.com/logcom/advance-article-pdf/doi/10.1093/logcom/exab006.

[8] C. Kaliszyk, J. Urban, H. Michalewski, M. Olšák, Reinforcement learning of theorem proving, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems 31, Curran Associates, Inc., 2018, pp. 8836–8847. URL: http://papers.nips.cc/paper/8098-reinforcement-learning-of-theorem-proving.pdf.

[9] K. Bansal, C. Szegedy, M. N. Rabe, S. M. Loos, V. Toman, Learning to reason in large theories without imitation, 2019. arXiv:1905.10501.

[10] V. Firoiu, E. Aygun, A. Anand, Z. Ahmed, X. Glorot, L. Orseau, L. Zhang, D. Precup, S. Mourad, Training a first-order theorem prover from synthetic data, 2021. arXiv:2103.03798.