Machine learning for automated theorem proving: an ML-side perspective

Boris Shminke

Expleo France

25 Mar 2024

Why am I talking about this

- working with data since 2009
- Mizar prize for young researchers in 2013
- applying ML in business since 2015
- 2020-2023 — PhD thesis on reinforcement learning for ATPs
- now back to R&D consulting at Expleo France

Outline

1 Benchmarking

2 Representation

3 Research software development

4 Conclusion

Outline

1 Benchmarking

2 Representation

3 Research software development

4 Conclusion

Competitive Programming

- contestant uploads a binary
- binary does I/O in a specified format
- there are time and RAM limitations

CASC rules about the output format

- "use of the TPTP format for proofs... is encouraged"
- "representative sample solutions must be emailed to the competition organizer"
- "systems must be sound"

How do we apply ML in business?

- ML is only one micro-service of a larger system
- we train our ML model separately
- we optimise the ML model for ML metrics
- we study correlation between ML metrics and business metrics

ML guidance for ATP

- everyone believes the ATP binary is sound
- ML gives advice on search decisions made by the ATP
- the more problems we solve, the better

ML benchmark example

ML model input

```
1.  ~man(X0) | mortal(X0) [input]
2.  man(socrates) [input]
3.  ~mortal(socrates) [input]
```

ML model output

```
[resolution 1,2]
```

ML for ATP guidance

- no ML-friendly benchmark publicly available
- a benchmark for one prover won't work for others (output format differs)

Why a purely ML benchmark?

- what metric to optimise
- how to represent data (feature extraction)
- which model/architecture to use
- what optimisation procedure to use
- how to produce predictions efficiently

Can we collect data from ATP usage?

- ITPs run ATPs routinely
- CASC runs ATPs en masse
- ATP developers run ATPs all the time

ML for ATP: a benchmark

- ATP researchers decide what to guide
- ML researchers do ML
- correlation between ML metrics and ATP metrics is studied together

Outline

1 Benchmarking

2 Representation

3 Research software development

4 Conclusion

Formulas are character sequences

- LLMs are great with character sequences
- LLMs will generate our proofs
- (when we have a large enough training dataset)

LLM is the king

- every image can be saved as an ASCII art file
- ASCII art file is a character sequence
- LLMs are great with character sequences
- we don't need computer vision anymore, do we?

Formal languages are not natural

- transformers work with sequences of tokens
- speech is a sequence of sounds
- formal languages are all about nesting
- can we define a token? Should we?

Formulas are graphs

- we can apply transformers to graphs (if we define tokens well)
- or transformers are only a special case of graph neural networks
- with which kind of graphs exactly?

Formula representations are not only for ATPs

- premise selection for ITPs
- SMT solvers use formulas too
- how about reasoners in ontologies
- or (functional) programming languages

Representation

- LLMs and transformers in general are not the final solution
- graphs (and which ones) might also be not
- formal language representation is not only for ATPs
- it could be easier to study representations with an ML benchmark

Outline

1 Benchmarking

2 Representation

3 Research software development

4 Conclusion

Code matters

- provers are not "supplementary material" no one will ever run
- much more info in the code than in published papers
- logician, programmer, and ML engineer are not the same person

When will we reuse TPTP parsers?

- can't create a prover without parsing input
- many abandoned TPTP parsers out there
- actively used parsers are undetachable from ATPs
- (with exceptions, e.g. LEO-III)

Every prover does inferences

- working with parsed formula-objects
- applying inference rules
- why not have a library of inference rules?
- (it might also help us to automatically evaluate provers)

ML guidance for ATPs is still painful

- papers without code
- code in experimental branches
- (with exceptions, e.g. iProver)

Provers are complex software

- adding ML inside won't make them less so
- using micro-service architecture could help
- reuse of common parts (parser, inferences) might help
- contributions from a wider open-source community might help
- (but we need CI, automated testing, and better documentation for that)

Outline

1 Benchmarking

2 Representation

3 Research software development

4 Conclusion

Main directions I see

- dedicated ML benchmarks
- universal formal language representations
- adopting open source software development practices

Thank you for your attention!

- questions are welcome
- let's discuss things while I'm in Vienna
- or write me an email at `boris.shminke@expleogroup.com`