Reinforcement Learning for Interactive Theorem Proving in HOL4

Minchao Wu[1], Michael Norrish[12], Christian Walder[12], and Amir Dezfouli[2]

[1] Research School of Computer Science
Australian National University, Canberra, ACT, Australia
[2] Data61, CSIRO, Canberra, ACT, Australia

We present an interface for reinforcement learning for interactive theorem proving in HOL4. The interface supports treating HOL4 as an interactive environment for agents to learn to prove theorems in a tactic style. We also describe in detail our reinforcement learning settings for the task, including the design of states, rewards and policy networks. We then give preliminary results demonstrating that theorem proving in HOL4 can be learned with our baseline approach of reinforcement learning using the interface.

Learning systems for interactive theorem proving have started to appear in recent years. Among them there are systems for special purposes such as premise selection [2][16] or algebraic rewriting [11]. There are supervised learning systems designed for general proof search such as TacticToe [8] for HOL4 [14] and CoqGym [18] for Coq [4]. There are also systems using deep reinforcement learning for general proof search such as DeepHOL [3] for HOL Light [9]. Our system is designed for general proof search in HOL4. Unlike TacticToe, which learns from human proof scripts without using deep learning, we use deep reinforcement learning to train policy networks to predict tactics as well as their arguments. Our system is also different from DeepHOL in the following aspects.

- The arguments of a tactic can be not only names of theorems, but also HOL4 terms. Like DeepHOL, predictions are made based on the embedded statements (i.e., expressions) of theorems, not their names.
- For tactics that can take more than one argument, an argument is predicted not only depending on the tactic and the context, but also the previously predicted arguments of the same tactic application. This is because some tactics, such as `simp` and `fs`, are sensitive to such dependence.
- The system does not assume a fixed set of tokens in advance. Once the agent is trained, it should be able to handle newly introduced definitions and theorems which are likely to contain new tokens invented by a user.

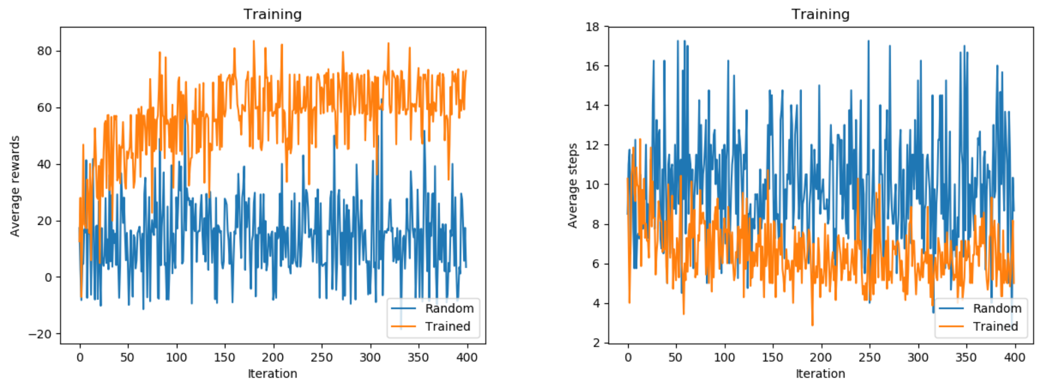
Another related implementation of deep reinforcement learning in HOL4 is given by Gauthier [7] recently. The implementation supports reinforcement learning inside HOL4 by implementing basic learning algorithms in standard ML. On the other hand, our interface supports interaction with HOL4 from within Python and manages proofs on the Python side. The interface is designed in a way that HOL4 theorem proving could be integrated as an actual Gym environment[5]. The environment provides information that can be directly processed by popular machine learning frameworks such as PyTorch [12] or TensorFlow [1].

Reinforcement learning formulation A proof attempt in HOL4 can be treated as a game. A state of the game is what we call a fringe. A fringe contains all the remaining goals of a proof attempt, along with their corresponding local context. If one thinks of proof search as a tree with edges being tactic applications and nodes being the resulting set of goals with their contexts, then the fringe is the union of the unexplored nodes at some stage. A game is won if the fringe becomes empty within a fixed number of timesteps. The action space can be arbitrarily large, as we consider a set of selected tactics as well as their arguments, which can possibly be all the definitions and theorems available in HOL4 or those provided by a user. During proof search, if a theorem is proved, then it is also added to the candidate pool from which an argument is chosen. We distinguish certain resulting states of a tactic application for reward shaping. An action is called ineffective if the tactic application does not change the goal nor its corresponding context. For an inapplicable or ineffective action, we penalize the agent by giving it a reward -2. If an action times out, then the agent receives a reward -1. If the agent managed to prove the main goal within a fixed number of timesteps, then we give it a positive reward that is sufficient to compensate the damage due to the penalization so that the accumulated reward of the episode would end up positive. In other situations, it receives a reward 0.

Policies Actions are predicted by a combination of three policy networks – a tactic policy for choosing a tactic, an argument policy for choosing a list of theorem names as the arguments of the tactic and a term policy for choosing a term if the tactic expects a HOL4 term as its argument. The tactic policy takes a state as an input, and returns a probability distribution π_{tactic} over the possible tactics. The agent then samples one tactic to apply according to π_{tactic}. The argument policy takes additionally the previously predicted argument and a hidden variable, and returns the scores s of the candidate theorems and a hidden variable h. An argument t is then chosen by sampling $\text{Softmax}(s)$. Then t and h are passed to the same policy again to predict next argument. The hidden variables are computed by a LSTM [10]. The term policy is similar to the argument policy, but the candidates are currently restricted to the tokens occurring in the goal being handled. In our basic settings, the predicted action is applied to the first element in the fringe by default. Backtracking is also not explicitly treated as an action in the basic settings, as it can be expected that the policy networks should learn to avoid unpromising applications by itself. However, more sophisticated approaches are always possible. For example, we can have an additional value network that scores the states for pruning unpromising actions.

Learning algorithms The policies are trained by policy gradient methods [15]. In our baseline approach, the policy networks are trained jointly using the REINFORCE [17] algorithm. We also describe the possibility of adding Monte-Carlo Tree Search [6] based on the learned policies as a policy improvement operator [13].

Preliminary Experiments We implement the baseline approach in PyTorch. Preliminary results are obtained based on the following settings. We train the agent to prove 10 theorems from the list theory of HOL4. Tactics allowed to be used in the proofs are `simp`, `fs`, `metis_tac`, `Induct_on`, `irule`, and `strip_tac`. For tactics that take theorems as arguments, we only allow the 56 definitions in the list theory to be chosen as the arguments. The length of the argument list is fixed to 5. Reuse of proved theorems is disabled. That is, the agent always tries to prove a theorem from scratch. For induction, the argument can be an arbitrary variable occurring in the goal. One iteration of training contains 10 episodes. Each episode is a proof attempt of one of the 10 theorems. If a theorem is proved, then the agent gains a reward of 100. Othewise, the rewards are as described in the above reinforcement learning formulation. The timeout limit for a single tactic is set to be 0.2 seconds. The timestep limit for a single



(a) Average total rewards received in each iteration. (b) Average steps to find a proof in each iteration.

Figure 1: Peformances in terms of total rewards and timesteps.

proof attempt is 20. We train the agent for 400 iterations and compare its performance against random rollouts with the same settings. It can be seen from Table 1 and Figure 1 that the agent is able to prove more theorems as the training goes on, and is guessing less to find a proof.

	average rewards	average steps	successful proofs	success rate
Overall	56.8	6.6	2771	69.2%
Last 100 episodes	65.7	5.9	75	75%
Random	14.2	10	1533	38.3%

Table 1: Performances of training and random rollouts on the same settings. Average steps refer to the number of timesteps needed to find a proof.

Improvements In our baseline approach, each formula in the fringe is given as a sequence of a finite number of tokens in Polish notation. The tree struture of a formula is not fully reflected in the representation which uses integer encoding, and the models are sequence-based. We plan to replace the current representation by more sophisticated ones such as learned embeddings using RNN as proposed in GamePad [11] or TNN for HOL4 terms as proposed in [7]. With better and deeper networks for both representation and policies, we hope that the performance of preliminary experiments generalizes to a larger scale. Other improvements include pre-training the policies on easy problems to accelerate training, or learning a supervised policy in advance [13] to help with proof exploration. We may also model backtracking by considering a proof graph as a state and allow the agent to choose fringes to work on.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike

Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Alexander A. Alemi, François Chollet, Niklas Een, Geoffrey Irving, Christian Szegedy, and Josef Urban. DeepMath - deep sequence models for premise selection. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2243–2251, USA, 2016. Curran Associates Inc.

[3] Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An environment for machine learning of higher order logic theorem proving. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 454–463, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[4] Pierre Boutillier, Stephane Glondu, Benjamin Grégoire, Hugo Herbelin, Pierre Letouzey, Pierre-Marie Pédrot, Yann Régis-Gianas, Matthieu Sozeau, Arnaud Spiwack, and Enrico Tassi. Coq 8.4 Reference Manual. Research report, Inria, July 2014. The Coq Development Team.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *CoRR*, abs/1606.01540, 2016.

[6] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[7] Thibault Gauthier. Deep reinforcement learning in HOL4. *CoRR*, abs/1910.11797, 2019.

[8] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Learning to reason with HOL4 tactics. *CoRR*, abs/1804.00595, 2018.

[9] John Harrison. HOL Light: An overview. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 60–66, Munich, Germany, 2009. Springer-Verlag.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[11] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. *CoRR*, abs/1806.00608, 2018.

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NeurIPS Autodiff Workshop*, 2017.

[13] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of GO without human knowledge. *Nature*, 550(7676):354, 2017.

[14] Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Ait Mohamed, César Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics*, pages 28–32, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[15] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[16] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2786–2796. Curran Associates, Inc., 2017.

[17] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforce-

ment learning. *Machine Learning*, 8(3):229–256, May 1992.

[18] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning*, 2019.