HOList: An Environment for Machine Learning of Higher-Order Theorem Proving (extended version)

Preprint, compiled April 9, 2019

Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, Christian Szegedy, and Stewart Wilcox

{kbk,smloos,mrabe,szegedy,stewbasic}@google.com
Google Research, Mountain View, California, USA

Abstract

We present an environment, benchmark, and deep learning driven automated theorem prover for higher-order logic. Higher-order interactive theorem provers enable the formalization of arbitrary mathematical theories and thereby present an interesting, open-ended challenge for deep learning. We provide an open-source framework based on the HOL Light theorem prover that can be used as a reinforcement learning environment. HOL Light comes with a broad coverage of basic mathematical theorems on calculus and the formal proof of the Kepler conjecture, from which we derive a challenging benchmark for automated reasoning. We also present a deep reinforcement learning driven automated theorem prover, DeepHOL, with strong initial results on this benchmark.

1 Introduction

Formalization of mathematics and the automated creation of new mathematical content is at the frontier of current AI techniques. Given the fundamental nature of mathematics and its importance for most scientific disciplines, the capability for high level formal mathematical reasoning is both an important practical task as well as one of the most challenging case studies in AI. However, traditional formal computer mathematics has been a fragmented domain, exploring various approaches for different logical foundations. This has led to a large number of incompatible theorem proving systems, which added extra challenges for AI researchers trying to push the limits of formal reasoning using machine learning.

Well-defined, large-scale benchmarks were instrumental for unifying disparate efforts in machine learning research: LibriSpeech [1] for speech recognition, the Netflix prize [2] for recommendation, ImageNet [3] for object recognition, MSCOCO [4] for object detection and segmentation, WMT [5] for machine translation, and SQuAD [6] for question answering, just to name a couple of examples. Benchmarks have fostered collaboration and competition and provide a means to measure progress, contributing significantly to accelerated progress and reproducible science.

This paper aims to provide such a benchmark and reinforcement learning environment for theorem proving. The long-term goal is to enable the automatic formalization of large theories, and hence we want to start with a theorem proving system that has a track-record of large-scale formalization efforts and includes a large corpus of foundational mathematics for benchmarking and learning. Our choice fell on HOL Light, the interactive theorem prover (ITP) in which the proof of the Kepler conjecture [7] has been formalized. The formalization of the proof of the Kepler conjecture has been a huge effort, taking over 20 person-years to complete, and required formalizing a significant part of arithmetic, linear algebra, and multivariate analysis. The resulting benchmark consists of 2199 definitions and 29462 theorems and lemmata, which capture a variety of interesting mathematics and should be a practical seed for new (auto-)formalization efforts.

To demonstrate the feasibility of the proposed learning task, we present an automated theorem prover powered by deep learning, called DeepHOL. Based on a simple solver architecture, DeepHOL learns how to prove theorems based on immitating human proofs and improves on itself using reinforcement learning. Therebby, DeepHOL achieves theorem proving capabilities that are comparable to much more complicated state-of-the-art automated theorem proving systems.

In our open-source release, we expose the APIs of our modular theorem prover. This simplifies the development of new provers significantly and allow researchers to focus on the machine learning aspects. Our release also includes *tactic argument selection*, which is our framework for premise selection for individual tactics.

The contributions of our work are the following:

- An instrumented, pre-packaged version of HOL Light that can be used as a large scale distributed environment of reinforcement learning for practical theorem proving using our new, well-defined, stable Python API. Our solution comes with optimized startup capabilities while allowing replay and strict verification of the produced proofs.
- Proof export and import capabilities that allow for managing large theories programmatically from the Python interface.
- A full-fledged, competitive automated neural theorem proving system that can automatize theorem proving in higher-order logic at tactic level directly.
- A large scale reinforcement learning system that was used for training our prover.
- Comparison of neural model architectures for theorem proving purposes.
- New, well-defined benchmarks on our HOL Light based environment to enable research and measuring progress of AI driven theorem proving in large theories.

The remainder of this work is organized as follows. We discuss related work in Section 2 before we discuss our theorem proving environment in Section 3. In Section 4 we describe how the organization of the benchmark. We present the DeepHOL automated theorem prover in Section 5 and discuss first experimental results for DeepHOL in Section 6 before we conclude in Section 7.

2 RELATED WORK

The earliest work of applying machine learning on reasoning in large theories is [8]. The most most similar works to ours are TacticToe [9] and GamePad [10]. TacticToe is the first published result on machine learning tackling higher-order theorem proving at a relatively large scale at tactic level [9]. Although TacticToe is a great success that came with significant improvements over previous automated theorem proving systems, they do not propose an easy to use benchmark or environment for machine learning researchers. TacticToe does not employ deep learning nor reinforcement learning. They rely on the HOL4 [11] system that has a significantly less theorems with more complex human proof scripts with a larger number of more elementary tactics.

GamePad has very similar objectives to ours [10]. They also provide an easy-to-use Python API for an interactive theorem prover, and they present test and training sets. They chose to base their system on Coq [12], an interactive theorem prover based on the calculus of inductive constructions. While enabling automatic code extraction, it comes with a much smaller coverage of fundamental mathematics. Even including the formalization of the Feit-Thompson theorem, their benchmark comprises only 1602 theorems and lemmas, while ours features 29462 theorems and lemmas. Besides presenting a much larger data set, we also demonstrate the feasibility of achieving state-of-the-art prover performance based on our data and environment by presenting a deep learning based theorem prover. We also report the results as theorem proving performance instead of proxy metrics.

Other interactive theorem provers we could have based a learning environment on include Mizar [13], Isabelle [14], HOL4 [11], and Lean [15]. The Mizar mathematical library is probably the most comprehensive formalization effort, but its declarative style makes it hard to employ proof search, and its source code is not freely available. Like Coq and HOL Light, also Isabelle [14] was used for major formalization efforts, such as the formalization of the seL4 microkernel [16]. We are not aware of a comprehensive coverage of fundamental mathematics in Isabelle, HOL4, or Lean.

In closely related work, Kaliszyk and Urban [17] translate from HOL Light and Flyspeck to automated theorem provers and SMT solvers, for which they learn a premise selector. In contrast to our work, they use neither deep learning nor reinforcement learning. Similar methods for premise selection on the HOL Light corpora were proposed in [18].

The first use of deep neural networks for large scale theorem proving was proposed in [19]. They have used convolutional networks for premise selection in large theories, particularly on Mizar mathematical library [13]. Those methods were used as a pre-selection for applying the first order logic automated theorem prover E [20]. We have reused several ideas from that paper, including some aspects of our neural network architecture and the hard negative mining methodology.

Whalen [21] proposed a purely deep reinforcement learning based solution for theorem proving for the Metamath prover [22]. This work was moderately successful, finding mostly proofs for very simple theorems, especially in propositional logic. On the other hand, Metamath is not considered to be a serious contender for large scale mathematical formalization work.

Loos et al. [23] proposed deep neural networks to augment theorem prover E [20] to rank given clauses during proof search. Here, we propose a neural prover written from scratch, relying solely on a small set of preexisting tactics and neural networks for all high level decisions.

Kaliszyk et al. [24] proposed a machine learning benchmark for higher-order logic reasoning based on the HOL Light corpus. It features a few static datasets and it remains unclear how performance of machine learning models on this dataset relates to real world prover performance. [25] demonstrated the viability of reinforcement learning with XGBoost and LIBLINEAR [26] on hand engineered features in first order logic context using leanCoP [27] on Mizar mathematical library [13].

Earlier works on employing (non-deep) machine learning for theorem proving in general and for reasoning in large theories include [28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44].

Recently, Wang et al. [45] proposed a premise selection method utilizing deep graphs embeddings.

3 ARCHITECTURE OF THE ENVIRONMENT

Here we describe the architecture of the evaluation and training environment. The goal of the environment is to enable artificial agents to interact with the HOL Light interactive theorem prover (ITP) in a replicable manner.

Background ITP terminology. In order to describe the changes to ITP we made, it is helpful to establish some terminology typically used. The task is to create a proof starting with the initial, top-level, *goal* (conjecture one is trying to prove). We assume a small, fixed number of so-called *tactics*, which are provided by an interactive theorem prover, and used as-is as black boxes. Some of these can use one or more previously proven theorems as parameters. The successful application of a tactic creates a new (possibly empty) set of subgoals to be proven. If a tactic is applied successfully and it does not create any new subgoals, then the subgoal to which the tactic was applied is successfully proven. On the other hand, if there are new subgoals, the subgoal will be considered proven when new subgoals are proven as well. If one thinks of the tree with subgoals as nodes, and tactic applications as outgoing edges, a *proof (graph)* then can be thought of as one where all the sinks are successfully proven by some tactic application. The tactic applications are sometimes referred to as *proof steps*.

3.1 Instrumentations to HOL Light

In order to create a stable, well-defined environment, we fix a particular version of HOL Light with a pre-selected subset of

tactics and a fixed library of basic theorems which are proved in one well-defined order. This is the ITP (interactive theorem prover) part of the environment which is written purely in OCaml with a few additional C++ functions for sandbox communication. Since it is technically non-trivial to find and build the exact correct set of libraries for this environment, we have prepackaged the whole compiled system as a docker image. It can be used as a reliable black box, communicated with using a stable API, and built upon for subsequent work. However, we have open sourced all the changes to the HOL Light system so that new modifications and forks are possible by third parties.

The prepackaged version we provide has the following additional instrumentations:

- Tracking and exporting of tactic applications and tactic arguments for existing proofs.
- Implementation of a fast startup for large theories by "cheating in" large libraries of statements quickly.
- A new stateless API to interact with a goal-stack for proof search.
- A theorem fingerprinting mechanism for fast communication, and stable referencing of theorems.

3.2 Proof Logging and Human Proofs

We want to utilize the existing human proofs for both training and evaluation purposes. To that effect, we have instrumented the most frequent tactics and the `prove` method in HOL Light with extra logging code. In addition, we have instrumented the `prove` method with extra logging an as well. The instrumented tactics additionally produce a proof log object that contains information on the tactic argument and also on the subgoal the tactic was applied to. These proof log objects can be used to recover the proof graph in case the proof only used tactics from the supported set of tactics. For each invocation of the `prove` function the proof log object is created and added to the list of existing proof objects. The order of the proof invocations is also maintained. If HOL Light is executed in a proof-dump mode, then all the proof logs are dumped into a file alongside the top level statement.

3.3 Fast Startup for Distributed Processing

In order to perform proof search utilizing all theorems prior to a given point in the theorem database, the HOL Light system must be initialized with all premises present in the HOL Light internal database. Given that this involves proving every theorem during the startup process of HOL Light, it ends up being very time consuming due to extensive proof checking of each potential premise. We measured it at 10-20 minutes to run proof checking for all of the *core* and *complex* libraries (see Section 4.4) containing the foundational theories. On the other hand, since the proof search itself is a very time consuming process (it can take multiple minutes for a few hundred tactic applications), We want a system that can be distributed efficiently over hundreds or thousands of workers. Therefore we want to minimize the startup time for each HOL Light instance in order to avoid adding a significant latency to the overall system. For this reason, we have added support for "cheating in" all high-level theorems from our corpora without actually checking their proofs. Of course, doing so is unsafe in theory, but "cheating in" is used only at proof search time. This cuts down the initialization time of the HOL Light system from over 10 minutes to mere seconds on each machine resulting in considerable savings, especially when the prover is to run on thousands of worker processes.

3.4 Proof checker

Any bug in the implementation of a theorem prover could make its reasoning unsound, rendering the whole formalization effort futile. For that reason, HOL Light is designed around a small trusted core of about 400 lines of OCaml code that builds proofs from few very basic rules. OCaml's type system guarantees that a theorem object can only be constructed by this trusted core. The rest of the HOL Light system can be seen as mere convenience features around that core.

Our "cheating in" of theorems described earlier for initialization during proof search bypasses this process for the sake of massive performance gains. To ensure that the proofs found during our proof search are correct nevertheless (and that our system does not merely learn to exploit a bug), we have implemented a *proof checker* that allows us to rerun a set of proofs in our proof log format with the same correctness guarantees of HOL Light's trusted core. Since the proofs that we want to check with the proof checker may depend on other theorems that could not be proven in the proof search, the proof checker hooks into the regular start-up of HOL Light and interleaves human-written proofs with the proofs to be checked. Thereby, the proof checker completely avoids the use of "cheating in" theorems. All results generated by our prover have been double-checked with the proof checker.

4 BENCHMARK

4.1 Training Examples

Our training examples are tuples of $(goal, tactic, arglist, negarglist)$. The *goal* is a theorem to be proven (represented as a string or sequence of tokens). The *tactic* is the ID of one of a preselected small set of tactics (currently consisting of 41 tactics), The *arglist* and *negarglist* are both list of theorems, each of which is represented as a string (sequence of tokens). *arglist* is the list of arguments that is passed to a tactic application, while *negarglist* is an optional list of arguments similar to the positive arguments, but not occurring in the argument list of the tactic application for the given *goal*. Additionally, there is a special argument signifying that the argument list should be empty.

However *negarglist* can also be empty, even if *arglist* is not. This is currently the case for all the examples generated from the human proof logs. *negarglist* theorems are typically collected during proof search from high scoring theorem argument proposals of the deep learning models in our reinforcement learning pipeline that are not actually necessary for the proof.

4.2 Tasks

Here we propose a few tasks that can be measured on these benchmarks.

	Definitions	Theorems	Proof states
core	240	2320	23512
complex	396	16623	509621
flyspeck	1563	10519	538540
all	2199	29462	1071673

Table 1: The three corpora of the benchmark. The core corpus contains the basic theorems that needed to define the tactics. While proofs of core theorems are useful for training, we omit them in validation, since some tactics assume those theorems. The complex corpus consists of theorems of complex calculus, and flyspeck contains most of the lemmas and theorems of the Kepler conjecture.

- On predicting the same tactic and tactic arguments that were employed in the human proof.
- On proving each of the subgoals in the proof log utilizing only some of the top level theorems preceding top level statement to which the subgoal belongs to.
- On proving each of the top level goals in the proof log utilizing only some of the top level theorems in the scope in the proof log.

4.3 Splits

Before training and evaluation, we have split the top level theorems into three subsets: training, validation and test set in a 60:20:20 ratio. Then each proof state (subgoal in the proof) is assigned to the same split as the theorem to which it belongs. The validation set can be used for continuous monitoring for proxy metrics of the model during training. Also the validation set is used occasionally to measure the end-to-end prover performance of the models during training. On the other hand, the test set must only be used extremely rarely for final assessment of a few models before publishing a paper alongside their validation set performance.

4.4 Datasets

Currently, we have three different kind of corpora as described in Table 1. When using the S-expression based format, the total number of distinct tokens in the datasets is 25402. Of these, many tokens are systematically generated by HOL Light (implicit types, generated variables). If we ignore these, we have around 2570 tokens.

5 DEEPHOL PROVER ARCHITECTURE

In this section, we describe the high-level architecture of our reference neural prover. The intelligence is fully learned without any hand-crafted features, and with very simple data preprocessing. We have no particular tweaks that were done for the particular logic or interactive theorem prover (ITP). All the engineering went into the neural network architecture, which is very generic, and into maintaining the proof search tree without any special regard for the particular ITP system. We use HOL Light and its logic (HOL) in our experiments, but our system is not specialized to it. We believe that our solution would also work with other goal-tactic based prover like Coq [12], HOL4 [11], or Lean [15]. Here we describe the details of our reference prover solution in detail.

5.1 Action Generator

From the previous section, it must be clear the most crucial part of our prover is the action generator that produces a list of tactic applications for a given subgoal. There are two decisions to be made:

- The tactic to be applied
- Creating an argument list for each of the tactics. (Comprised of a list of theorems)

Each subgoal is comprised of a list of hypotheses and a conclusion and in our current system based on HOL Light, we base our decision of tactic application on the conclusion alone. The conclusion is passed to the action generator as a S-expression of the HOL term. As noted earlier, for some tactics it is possible to pass in arbitrary terms (formulas) as parameters. Our current system never creates any term from scratch, only ranking previously proven statements. In a future section, we describe the precise details of the neural architecture that we used for predicting the tactics and their arguments.

5.2 Proof search graph

We now elaborate on how we capture and maintain state of the proof search, and from which it is possible to detect when a proof for the original goal is available. During our proof search, we maintain a search graph, in which each subgoal can have multiple alternative tactic applications, each of which might result in multiple subgoals. Our current simple search process is just a breadth first search, in which we limit the number of tactic application expansions (elaborated in following subsection). Given the subgoal and the allowed set of premises, the action generator produces a list of possible tactic applications sorted by their score, where the size of this list is limited. Then these tactics are applied until a preset number of successful tactic applications is reached. Tactic applications resulting in the input subgoal are discarded in the process. The resulting new subgoals are added to the search graph and to the queue of tactic applications to be processed. Whenever a tactic application closes a subgoal, this information is traced back to the parent subgoals and each alternate tactic application (and its whole sub-branch) is marked as closed is discarded from the queue to be processed. If during this recursive process the root node is reached, then the proof is closed and the proof process stops. Another possibility is that all tactic applications fail or fail to change any given subgoal. In this case, we assume that subgoal cannot be proved. This would mean that all sibling subgoals become superfluous as well. Additionally the information might propagate up, if all alternate subgoals fail to prove. Again, our system propagates this information automatically and incrementally as far as necessary.

5.3 Proof Search

Our proof search is simple breadth first search that considers proof branches that are in the top-k successful tactic applications that change the current subgoal. At every step, we take the topmost element of the subgoal queue and try to prove it

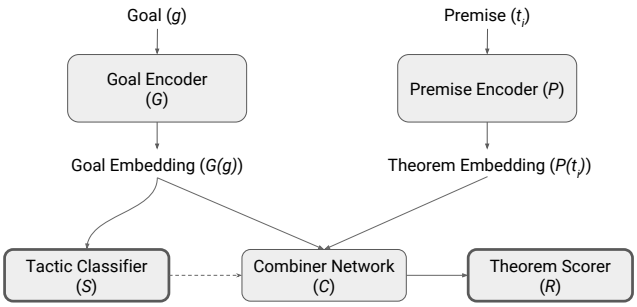

Figure 1: Two-tower neural architecture for ranking actions.

or expand it into new subgoals. One important feature of our system is subgoal sharing: every newly created subgoal is compared to all existing subgoals and a subgoal is shared whenever they are identical. This is crucial, otherwise the search process could end up oscillating between two formulas by rewriting the same subterm back and forth using the same equation. Once a subgoal is newly shared, previously stored information about subgoals being closed or ignored should be propagated through the search graph. Whenever a complete proof is found for the top level goal, the proof search is stopped and the whole proof search graph is serialized and stored as the result. Also, the proof search finishes if the search graph reaches a prescribed limit on the number of subgoals or the proof search times out.

5.4 Neural Architectures

For the generation and ranking of actions, we use a deep, two-tower neural network depicted in Figure 1. It has two separate prediction heads S and R. Goal tower G computes an embedding $G(g)$ of the current goal g and infers a scoring vector $S(G(g))$ for the fixed set of tactics where the tactic classifier S is a linear layer producing logits of a softmax classifier. The premise tower P computes a fixed size embedding $P(t_i)$ of all possible tactic arguments t_i in the scope of the goal to be proved. The ranking of the premises is performed by a combiner network C that takes the concatenation of the goal embedding, the premise embedding and possibly that of the tactic T_j to be applied: $r(t_i) = C(G(g), P(t_i), T_j)$, where $r(t_i)$ is the score of theorem t_i for its being a useful tactic argument for transforming the current goal g towards a closed proof. We have also tried the unconditioned setup, in which the ranking of the tactic arguments is independent of that of the tactic to be applied, that is $r(t_i) = C(G(g), P(t_i))$. In essence, we propose a hybrid architecture that both predicts the correct tactic to be applied, as well as rank the premise parameters required for meaningful application of the tactics.

5.5 Supervised Learning

In the supervised learning setup, we use the human proof logs to train a model upfront. For a description of our splitting the theorems and training examples into subsets, see Section 4.

We always report both validation and test set performance for the final result to verify that we did not over-fit on the validation set. Continuous measurements and ablation analyses are reported only on the validation or training set.

5.6 Reinforcement Learning Loop

In the reinforcement learning loop, we have both a trainer and multiple provers running continuously.

The training is (optionally) seeded with training examples from existing (human/generated) proof logs. Then, we run the neural prover in *rounds*, in each round trying to prove a random sample of theorems in the training set. Training examples extracted from successful proof logs of each round of our neural prover are mixed in continuously. We allow for weighing examples of more recent rounds of our prover (*fresh* examples) differently from older rounds (*historical* examples) during the training process.

To summarize, our loop works with the following four kinds of training example pools:

1. (optional) Human training examples as seed.
2. (optional) Inherited computer generated examples as seed: in addition to using human training examples as seed, examples generated during any previous experiments with our prover can also be used as seed. In our current experiments, we used examples that were generated by a prover that was run on the whole training set utilizing a model that was trained in purely supervised manner.
3. Fresh generated loop examples (examples that were produced in the last k rounds, where k is a user-settable parameter).
4. Historical training loop examples (examples that were produced in all but the last k rounds).

During training, examples from each pool are selected according a prescribed split ratio. This means that the ratio of different kinds of examples in each batch is consistent and does not shift as more and more examples generated by the loop. Most importantly, it also ensures that examples from freshly constructed new proofs show up quickly and deterministically during the training process. Note that although we can make use of human and inherited proof traces, the system in theory can learn without any supervision or initial seed data. However our preliminary experiments have shown that in its current form it learns significantly inferior models compared to those that were seeded with human proof log examples and from proof traces produced using prover using models trained in a supervised manner.

5.6.1 Proof Pruning

In order to obtain high quality training data for premise selection (i.e. tactic argument prediction), we *prune* the predicted parameter list. For all tactics that take a list of theorems as an argument, our current implementation generates a list of fixed length. For successful tactic applications, we then iterate over the arguments in reverse score order and greedily omit those arguments that do not change the outcome of the tactic application. While a non-greedy approach might yield even shorter argument lists, it would also take longer to compute. In practice, our approach produces very short argument lists with minimal effort. Removed parameters are stored as "hard negatives" and utilized during training.

Description	Proof success
ASM_MESON_TAC	6.1%
ASM_MESON_TAC + argument selection	9.2%
1 Convolutional layer	24.1%
2 Convolutional layers	17.1%
WaveNet	24.0%
Loop	36.3%
Trained on loop output	36.8%
Loop tactic dependent	**38.9**%
Loop on subgoals	34.6%

Table 2: Percentage of theorems closed using various methods on the validation set of the complex corpus comprising on 3217 theorems. First two lines are trivial baselines that call HOL Light's builtin first order theorem prover with and without utilizing our argument selection model. Middle section shows results of models trained in a supervised scenario on human proof logs, where the last line of that section utilizes synthetic proofs as well. The last four lines report results using our reinforcement learning loops. Test set performance is evaluated on two models that yielded best result on the validation set in their category.

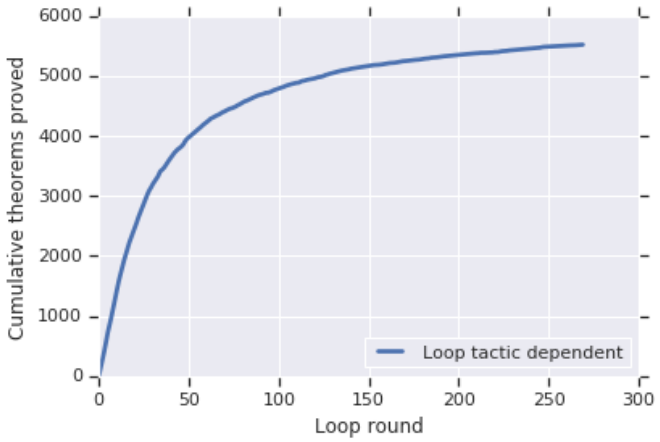

Figure 2: This figure presents the cumulative number of proofs closed by the tactic dependent loop. The total number of theorems in the training set is 10199.

Maximum number of top tactics explored	[6, 16]
Maximum *successful* tactic applications	[3, 6]
Number of selected tactic arguments	[1, 32]

Table 3: Randomized proof search parameters and their ranges.

6 RESULTS AND COMPARISONS

In this section, we first present several baseline results based on immitation (i.e. fully supervised) learning. Then we come to our reinforcement learning results using the same WaveNet [46] based encoder architecture, but with three different training methodologies.

6.1 *Model Training Hyperparameters*

All models were trained with Adam optimizer [47] and exponentially decreasing learning rate starting at 0.0001 with decay rate 0.98 at every 100000 steps. For evaluation, we use moving exponential parameter averaging at a rate of 0.9999 per step [48, 49]. First, we established trivial baselines by running the builtin first-order theorem prover `ASM_MESON_TAC` on each theorem on the dataset with empty argument list and with an argument list predicted with our baseline WaveNet model. Next, we compare the performance of various simple multi-layer convolutional networks and that of the WaveNet style architecture. Finally, we report our reinforcement learning experiments on the complex analysis corpus.

6.2 *Comparison of Model Architectures*

We trained and evaluated a large number of convolutional networks with up to four layers, rectified linear application, stride 2 from the second layer, patch size in $\{3, 5, 7\}$, and filter size in $\{128, 256, 512, 1024, 2048\}$. During our experiments, we looked at the following proxy metrics:

1. Accuracy of tactic prediction out of the 41 possible tactics. (Ranging between 38% and 42% for most models.)
2. Success rate of selecting a positive tactic argument over a randomly selected negative argument. (Around 1% error rate).

In addition, we trained a decoder with two WaveNet [46] style towers with two blocks of four layers each, and where the number of filters in each block was either 128 or 256. What we have found is that all our simple convolutional networks were inferior on both proxy metrics: tactics prediction accuracy and relative argument ranking. The only exceptions were one layer neural networks with large number of filters (1024) and patch size 7, that seemed to outperform our baseline WaveNet model on argument ranking, but was significantly weaker on tactic prediction. Our best simple model for tactic prediction was a two layers neural network with 1024 filters on each layer, patch size 7 and with stride 2 for second layer. Here we present various results on the complex analysis corpus via trivial baselines, supervised training, and using reinforcement learning. Our final prover performance numbers are summarized in Table 2.

6.3 *Reinforcement Learning*

In our reinforcement learning set up, the model training runs on single GPU, while theorem proving is performed in a distributed manner: we attempt to prove 2000 randomly selected theorems from the training set of union of the complex and core corpora in every round. At the start of each round, we fetch the latest trained model checkpoint and precompute the theorem argument embedding for each theorem in the complex and core libraries. This precomputation greatly accelerates the ranking of the tactic arguments. The proof search is distributed over 1000 cores and we set a computation limit of 100 explored proof states and a total timeout of 300 seconds. Each individual tactic application has a timeout of 5 seconds. Additionally, for each example, we pick prover options uniformly in the ranges described by Table 3.

The reason we use randomized parameters is to increase the diversity of the generated proofs, since we have only about 12

Name	Theorems proved (% of training set)
Loop	5679 (55.7%)
Loop tactic dependent	5518 (54.1%)
Loop on subgoals	1988 (19.5%)
Union	5919 (58.0%)

Table 4: Total count of proofs found by each loop.

Figure 3: Percentage of theorems proved in each round of the loop. Each round samples 2000 theorems from the training set.

thousand top level statements to pick from. This also increases the chances finding a proof at all for harder statements, by increasing the search space.

Given the high computational cost of running the reinforcement learning loop, we have only tried a couple of variants. Each of our these experiments use the same version of WaveNet [46] architecture (with 128 filters in each layer). In our first loop experiment, "Loop", we trained a loop with tactic independent argument selection. That is, the tactic argument ranking was independent of the tactic chosen, and we pick only top level theorems to be proved by proof search in the reinforcement learning scenario. Alongside our first loop, we trained a separate model "Trained on loop output" that was not used in the loop for proof search guidance, but did benefit from a curriculum-style learning, since it trained in parallel to the loop. In future work, we hope to run additional experiments to explore the contributions of curricula learning on synthetic data vs. training the model in a reinforcement loop. In our second loop experiment, "Loop tactic dependent", we have trained a model in which the arguments ranking depends on the selected tactic. In our third loop experiment, "Loop on subgoals", the proof search can pick from any of the internal proof states from the training set of the joined core+complex corpus. This means, that we expected a bigger variety of theorems to be generated during proof search. Performance of each loop's final checkpoint on the validation set is presented in Table 2. We also ran the final checkpoint of the "Loop" on a sample of 2000 proofs from the *flyspeck* dataset; we closed 752 (37.6%) of these proofs automatically.

While it was too computationally expensive to track the validation performance on every round of the loop, we did record the performance on the training set. In Fig. 2, we show the cumulative number of proofs closed by the tactic dependent loop at each round. Recall that in each round we sample theorems from the training set and use the most recent checkpoint to guide the proof search. In Fig. 3, we show the percentage of the sampled theorems that are proved in each round.

7 CONCLUSION

We presented a machine learning oriented open source environment for higher-order theorem proving as well as a neural network based automated prover, trained on a large-scale reinforcement learning system. We also suggest a benchmark for machine reasoning in higher-order logic on a relatively large and practically relevant corpus of theorems with varying complexity. Our benchmark includes purely neural network based baselines, which demonstrate strong automated reasoning capabilities, including premise selection from a large number of theorems. We hope that our initial effort fosters collaboration and paves the way for strong and practical AI systems that can learn to reason efficiently in large formal theories.

REFERENCES

[1] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5206–5210. IEEE, 2015.

[2] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA, 2007.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[4] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[5] Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, et al. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the ninth workshop on statistical machine translation*, pages 12–58, 2014.

[6] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[7] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. A formal proof of the kepler conjecture. In *Forum of Mathematics, Pi*, volume 5. Cambridge University Press, 2017.

[8] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. Malarea sg1-machine learner for automated reasoning with semantic guidance. In *International Joint Conference on Automated Reasoning*, pages 441–456. Springer, 2008.

[9] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Tactictoe: Learning to reason with hol4 tactics. In *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46, pages 125–143, 2017.

[10] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. *arXiv preprint arXiv:1806.00608*, 2018.

[11] Konrad Slind and Michael Norrish. A brief overview of hol4. In *International Conference on Theorem Proving in Higher Order Logics*, pages 28–32. Springer, 2008.

[12] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[13] Mizar. The Mizar Mathematical Library. URL http://mizar.org. Accessed: 2018/01/18.

[14] Makarius Wenzel, Lawrence C. Paulson, and Tobias Nipkow. The isabelle framework. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *Lecture Notes in Computer Science*, pages 33–38. Springer, 2008.

[15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *International Conference on Automated Deduction*, pages 378–388. Springer, 2015.

[16] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-752-3. doi: 10.1145/1629575.1629596. URL http://doi.acm.org/10.1145/1629575.1629596.

[17] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with flyspeck. *Journal of Automated Reasoning*, 53(2):173–213, 2014.

[18] Cezary Kaliszyk and Josef Urban. Initial experiments with external provers and premise selection on hol light corpora. 2012.

[19] Alexander A Alemi, François Chollet, Geoffrey Irving, Niklas Eén, Christian Szegedy, and Josef Urban. Deepmath-deep sequence models for premise selection. In *Advances in Neural Information Processing Systems*, pages 2235–2243, 2016.

[20] Stephan Schulz. E - A Brainiac Theorem Prover. *AI Commun.*, 15(2-3):111–126, 2002.

[21] Daniel Whalen. Holophrasm: a neural automated theorem prover for higher-order logic. *arXiv preprint arXiv:1608.02644*, 2016.

[22] Norman Megill. Metamath: A computer language for pure mathematics. 1997.

[23] Sarah Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. *arXiv preprint arXiv:1701.06972*, 2017.

[24] Cezary Kaliszyk, François Chollet, and Christian Szegedy. Holstep: A machine learning dataset for higher-order logic theorem proving. *arXiv preprint arXiv:1703.00426*, 2017.

[25] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olšák. Reinforcement learning of theorem proving. *arXiv preprint arXiv:1805.07563*, 2018.

[26] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

[27] Jens Otten and Wolfgang Bibel. leanCoP: lean connection-based theorem proving. *J. Symb. Comput.*, 36(1-2):139–161, 2003.

[28] Stephan Schulz. *Learning search control knowledge for equational deduction*, volume 230 of *DISKI*. Infix Akademische Verlagsgesellschaft, 2000. ISBN 978-3-89838-230-4.

[29] Hazel Duncan, A Bundy, J Levine, A Storkey, and M Pollet. The use of data-mining for the automatic formation of tactics. 2004.

[30] Josef Urban, Jiří Vyskočil, and Petr Štěpánek. Malecop machine learning connection prover. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 263–277. Springer, 2011.

[31] Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In *International Joint Conference on Automated Reasoning*, pages 378–392. Springer, 2012.

[32] Cezary Kaliszyk and Josef Urban. Stronger automation for flyspeck by feature weighting and strategy evolution. 2013.

[33] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. Mash: machine learning for sledgehammer. In *International Conference on Interactive Theorem Proving*, pages 35–50. Springer, 2013.

[34] Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *Journal of Automated Reasoning*, 52(2):191–213, 2014.

[35] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. Machine learning for first-order theorem proving. *J. Autom. Reasoning*, pages 1–32, 2014. ISSN 0168-7433. doi: 10.1007/s10817-014-9301-5. URL http://dx.doi.org/10.1007/s10817-014-9301-5.

[36] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Machine learner for automated reasoning 0.4 and 0.5. *arXiv preprint arXiv:1402.2359*, 2014.

[37] Cezary Kaliszyk, Lionel Mamane, and Josef Urban. Machine learning of coq proof guidance: First experiments. *arXiv preprint arXiv:1410.5467*, 2014.

[38] Michael Färber and Cezary Kaliszyk. Random forests for premise selection. In *International Symposium on Frontiers of Combining Systems*, pages 325–340. Springer, 2015.

[39] Cezary Kaliszyk and Josef Urban. Mizar 40 for mizar 40. *Journal of Automated Reasoning*, 55(3):245–256, 2015.

[40] Cezary Kaliszyk, Josef Urban, and Jirí Vyskocil. Efficient semantic features for automated reasoning over large theories. In *IJCAI*, 2015.

[41] Cezary Kaliszyk and Josef Urban. Femalecop: Fairly efficient machine learning connection prover. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 88–96. Springer, 2015.

[42] Cezary Kaliszyk and Josef Urban. Learning-assisted theorem proving with millions of lemmas. *Journal of symbolic computation*, 69:109–128, 2015.

[43] Thibault Gauthier and Cezary Kaliszyk. Premise selection and external provers for hol4. In *Proceedings of the 2015 Conference on Certified Programs and Proofs*, pages 49–57. ACM, 2015.

[44] Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for isabelle/hol. *Journal of Automated Reasoning*, 57(3):219–244, 2016.

[45] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In *Advances in Neural Information Processing Systems*, pages 2786–2796, 2017.

[46] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[47] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[48] Boris Teodorovich Polyak. A new method of stochastic approximation type. *Avtomatika i telemekhanika*, 7:98–107, 1990.

[49] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.