Extending HOList for Rapid Prototyping and Development

Jordan Starkey
jds95@knights.ucf.edu
University of Central Florida
Orlando, Florida

Tyler McFadden
neruelin@knights.ucf.edu
University of Central Florida
Orlando, Florida

Aaron Hadley
Aaron.Hadley@knights.ucf.edu
University of Central Florida
Orlando, Florida

Sumit Jha
sumit.jha@ucf.edu
University of Central Florida
Orlando, Florida

ABSTRACT

Higher-order interactive theorem provers enable us to drive auto-formalization efforts for mathematical problems. State of the art theorem provers rely on advanced tactics such as resolution or analytic tableaux. Unfortunately, today's best automatic theorem provers are weaker than their human counterparts and none have been able to formalize the "top 100" mathematical theorems [1]. Machine learning trained on large formal proof corpora [3] could help to eliminate this current bottleneck by being applied to various problems in the proof search process.

Recent efforts have yielded success in the application of machine learning to automated theorem proving, but the domain remains fragmented. HOList utilizes the HOL-light proof assistant to provide an extensible, open-source reinforcement learning environment.[3] GamePad provides a learning environment for the Coq proof assistant.[5] We present an accessible tutorial for configuring and using HOList, a Jupyter notebook environment for rapid prototyping within DeepHOL.

CCS CONCEPTS

• **Theory of computation → Automated reasoning**; • **Computing methodologies → Neural networks**; **Markov decision processes**.

KEYWORDS

theorem proving, neural networks, machine learning, deep learning, artificial intelligence, higher order logic

ACM Reference Format:
Jordan Starkey, Tyler McFadden, Aaron Hadley, and Sumit Jha. . Extending HOList for Rapid Prototyping and Development. In . ACM, New York, NY, USA, 8 pages.

1 INTRODUCTION

The field of automated theorem proving aims to solve any formally stated mathematical problem using human-guided heuristics. Automated reasoning over such mathematical proofs is a key aspect of general intelligence. With auto formalization, comes a broad set of applications (automated programming, software verification) and the pipe-dream of general intelligence. Reasoning and abstracting over mathematical formulae are believed to be the next steps in deep learning research. [?]

Its unfortunate bottleneck is within the human expertise required as it can be only as good as the current domain knowledge and the overhead for setting up software to do this is nontrivial. Automated Theorem Provers(ATP) require a high level of natural language understanding to formalize mathematical conjectures. Although these provers have robust strategies, such as resolution refutation or superposition calculus, they do not scale well due to the exponential search space when applied to larger conjectures.

The goal of applying deep learning to formal reasoning is to use the super-human pattern matching capabilities of these agents to augment explicit search and formal logic systems. Our senior design project implements the work done at DeepMath, a Google Research project aimed at applying an intuition model (symbolic artificial intelligence) to automated theorem proving. This is the first application of deep learning to theorem proving where the neural network models are trained on pairs of first-order logic axioms and conjectures to determine which axiom is most likely to be relevant in constructing an automated proof by the prover

1.1 Approach

Machine learning augmented for automatic and interactive theorem provers has been applied to the following three tasks: internal guidance, premise selection, and strategy selection.

The task of internal guidance requires machine learning methods to aid in the inference steps of the automated theorem prover. The benefit of this is that restricting the possible inference steps to a more conducive action set greatly reduced the proof search space. [] applied this method to first-order tableaux with . through probabilistic analysis

A second theorem proving task is a multi-label classification problem that is the focus of the HOList team. The formal problem statement of the premise selection task was first described as follows: Given a formal corpus of facts and proofs expressed in an ATP-compatible format, our task is:

Definition. *Given a large set of premises P, an ATP system A with given resource limits, and a new conjecture C, predict those premises from P that will most likely lead to an automatically constructed proof of C by A.*

This task is especially compelling as it formalizes theorem proving as a search problem; one where we start at the root (given facts) and traverse the tree (possible tactic deductions)until we reach the first root (containing our conclusion). This is crucial for

Finally, the importance of efficient strategy selection is imperative as many modern ATPs such as E and Vampire come complete with command languages that allow the user to control different heuristics including ordering, weighting functions, etc. The informal problem statement of the strategy selection task is described as:

Definition. Given a problem p in our problem set P, we define a strategy(s) with s in our parameter space S that can solve problem p.

This parameter space can be very large. The ATP E allows for over 10^{17} search strategies. To simplify the parameter space, ATPs usually only consider specific strategies that are successful and robust. To figure out the best pre-selected strategy to apply, we must then characterize the problems. This is known as picking a $feature$ set. Ideal features are computationally fast and are expressive so that they can handle different proofs. Examples of these features include number of clauses, negative literals, etc.

2 RELATED WORK

2.1 HOList

Our work is based on HOList, an open-source reinforcement learning environment developed by Bansal et. al. as a collaborative project between Google Research and several Universities. HOList consists of two parts. A modified version of HOL-light, an interactive theorem prover, and DeepHOL, a theorem prover driven by deep-learning. The two parts communicate via a GRPC API, making the system easy extensible. The HOL-light website provides instructions for installation via docker images for both the DeepHOL prover and HOL-light proof assistant. Also provided are benchmarks for machine learning techniques. [2]

2.2 GamePad

GamePad is an Reinforcement Learning environment published by a collaboration between researchers from UC Berkeley and OpenAI that uses the Coq Proof Assistant as its proving engine. It provides an interface that has the ability to build machine verifiable proofs in a step-by-step manner. The steps in these proofs are in the same tactics-based style as HOL making the GamePad system appear to be a promising target for integration with the gRPC protocol defined in HOList.

3 DEEPHOL TUTORIAL

Here we provide an introductory tutorial for setting up and configuring HOList. The official documentation

can be found at
https://deephol.org. A version of our tutorial can be found on
github at github.com/aahadley/deepmath-jupyter, or in the Jupyter notebook located at
github.com/aahadley/deepmath-jupyter/blob/master/HOLJup.ipynb.

3.1 Installing from source

3.1.1 *Dependencies.*

The required dependencies for running and installing HOList are:

- Ubuntu Linux*
- Python 2.7, Python 3.5+, Python 3.5+-dev
- docker
- wget
- unzip

**HOList does not explicitly require Ubuntu, but has not been tested on other systems*

3.1.2 *Virtual Environment.*

We highly recommend running in a virtual environment and using an environment manager. Instructions will be given for Anaconda and Python 3.75.

Create a conda environment with packages *h5py, six, numpy, scipy, wheel, mock, pyfarmhash, grpcio, grpcio-tools,*
keras_applications, keras_preprocessing

```
$ conda create --name deephol
$ conda activate deephol
$ pip3 install h5py six numpy scipy wheel
    mock pyfarmhash grpcio
$ pip3 install keras_applications==1.0.6
    keras_preprocessing==1.0.5 --no-deps
```

3.1.3 *Get Files.*

Clone the necessary repositories and descend into the deepmath folder.

```
$ git clone https://github.com/tensorflow/
    deepmath.git
$ git clone https://github.com/brain-
    research/hol-light.git
$ cd deepmath-jupyter
```

3.1.4 *Update TensorFlow Submodules.*

```
$ cd deepmath
$ git submodule update --init
```

3.1.5 *Bazel.*

Install Bazel, a build manager for TensorFlow. We need to make sure to use the 0.21 release, so we need to install from source.

```
$ wget https://github.com/bazelbuild/bazel/
    releases/download/0.21.0/bazel-0.21.0-
    installer-linux-x86_64.sh
$ chmod 777 bazel-0.21.0-installer-linux-
    x86_64.sh
$ bash bazel-0.21.0-installer-linux-x86_64.
    sh --prefix=$HOME/bazel --user
$ PATH=$HOME/bazel/bin:$PATH
```

3.1.6 *Configure Tensorflow.*

Navigate to the `tensorflow/` directory and configure TensorFlow. You can make changes to this step depending on your own configuration needs.

```
$ cd deepmath/tensorflow
$ TF_IGNORE_MAX_BAZEL_VERSION=1
    TF_NEED_OPENCL_SYCL=0
    TF_NEED_COMPUTECPP=1   TF_NEED_ROCM=0
    TF_NEED_CUDA=0   TF_ENABLE_XLA=0
    TF_DOWNLOAD_CLANG=0   TF_NEED_MPI=0
    TF_SET_ANDROID_WORKSPACE=0 CC_OPT_FLAGS=
    "-march=native␣-Wno-sign-compare"  ./
    configure
```

Now set the `PYTHON_BIN_PATH` variable.

```
$ export PYTHON_BIN_PATH=$(which python)
```

3.1.7 *Build with Bazel.*

Now we can build TensorFlow with Bazel. This step will take 30-60 minutes and may fail if you have less than 8GB of memory available.

```
$ cd ..
$ bazel build -c opt //deepmath/deephol:main
     --define grpc_no_ares=true --
    python_path=$PYTHON_BIN_PATH
```

3.2 Installing HOL-Light container

A modified version of HOL-light is provided by Google as a docker container for ease of use. [2]

First, create a docker network, then download and run the container. It is also a good idea to create an environment variable for the container's ip address.

```
$ docker network create holist_net
$ docker run -d --network=holist_net --name=
    holist gcr.io/deepmath/hol-light
$ export HNET_IP="$(docker␣inspect␣--format
    ='{{range␣.NetworkSettings.Networks}}{{.
    IPAddress}}{{end}}'␣holist)"
```

This command will stop the container.

```
$ docker stop holist && docker rm holist &&
    docker network rm holist_net
```

Google also put their neural proving agent, DeepHOL, in docker container. This lets each of the two components run as a black box. The HOList docker container acts as a server and the DeepHOL docker container acts as a client. The client makes queries to the server which responds (via gRPC which is built on top of protoBuf).

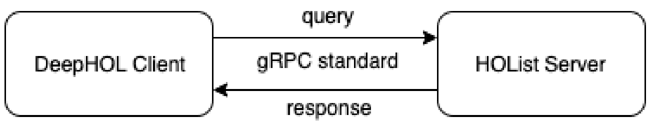

Figure 1: A high level diagram of the Docker and gRPC interface adapted from https://github.com/jasonrute/holist-communication-example

The protoBuf found in the deepmath repositroy at `/deepmath/proof_assistant/proof_assistant.proto` describes the three types of requests one can make of the HOList server.

```
service ProofAssistantService {
  // Apply a tactic to a goal, potentially
      generating new subgoals.
  rpc ApplyTactic(ApplyTacticRequest)
      returns (ApplyTacticResponse) {}

  // Verify that a sequence of tactics
      proves a goal using the proof
      assistant.
  rpc VerifyProof(VerifyProofRequest)
      returns (VerifyProofResponse) {}

  // Register a new theorem with the proof
      assistant and verify/generate
  // its fingerprint. The theorem will be
      assumed as one that can be used
  // as a valid premise for later proofs.
  rpc RegisterTheorem(RegisterTheoremRequest
      )
      returns (RegisterTheoremResponse) {}
}
```

This structure opens up the possibility for further modularity. In theory, we can simply dockerize new prover agents and proof assistant clients by following the schema layed out by the HOList system.

We would like to thank Dr. Jason Rute for his assistance in understanding the HOList API.

3.3 Running DeepHol

Before running, it is necessary to have a configuration file, theorem database, and model checkpoints (if applicable). A sample of these files can be found at https://storage.googleapis.com/deepmath/deephol.zip. Download these files and store them wherever is convenient.
We chose `deepmath-jupyter/data`

```
$ wget https://storage.googleapis.com/
    deepmath/deephol.zip -O /tmp/deephol.zip
$ unzip /tmp/deephol.zip -d ./data
```

Now DeepHOL can be run with:

```
$ python bazel-bin/deepmath/deephol/main --
    prover_options=data/configuration/
    prover_options.textpb --output=data/
    proof_logs.textpbs --
    proof_assistant_server_address=$HNET_IP
    :2000
```

After deephol runs, it will generate a proof log at `proof_logs.textpbs`. These proofs need to be checked with HOL-Light. To do so, copy the output file to the hol-light directory, build the HOL-light container, then run hol-light.

```
$ python bazel-bin/deepmath/deephol/
    utilities/proof_checker \
--theorem_database=/data/theorem_database_v1
    .1.textpb \
--proof_logs=data/proof_logs.textpbs \
--out_file=data/synthetic_proofs.ml

$ cp data/proof_logs.textpbs ../hol-light/
    proof_logs.textpbs
```

```
$ cd ..
$ docker build -f hol-light/
    Dockerfile_check_proofs --ulimit stack
    =1000000000 --tag check_proofs hol-light
    /
$ docker run check_proofs
```

3.4 Configuration

Configuration options can be found in `deepmath/data/configuration/prover_options.textpb`.
Available configuration options are determined by `deepmath/deephol/deephol.proto`.
Whenever changes are made to `deephol.proto`, it must be compiled with:

```
$ python -m grpc_tools.protoc -I. --
    python_out=bazel-genfiles/deepmath/
    deephol/ deepmath/deephol/deephol.proto
```

3.5 Models

The purpose of this task is to extend the baseline models for Holstep. The long-term aim is to build a neural network useful in automated theorem proving.

The current code provides just step classification but it can be extended for any other tasks including formula generation (decoding)

Usage. Prerequesites:

- Python2.7
- Tensorflow==1.14.0
- deepmath repository (https://github.com/tensorflow/deepmath.git)
- scikit-learn
- keras
- HolStep dataset (http://cl-informatik.uibk.ac.at/cek/holstep/)

From `deepmath-jupyter/data`

```
$ python main.py \
--model_name=cnn_2x_siamese \
--task_name=conditioned_classification \
--logdir=experiments/
    cnn_2x_siamese_experiment \
--source_dir=~/holstep \
--run_type=single \
--model_type=keras}
```

`model_name` is the name of a model (see which models are available in `conditioned_classification\_models.py` and `unconditioned_classification_models.py`). `task\name` may be either `conditioned_classification` or `unconditioned_classification`. `run_type` maybe be either single or `search`. The latter performs a grid search with defaulted hyperparameter lists. These can be modified in the source code. `model_type` is either keras, tf, or sklearn. Non-neural models are implemented with Tensorflow and Sklearn.

3.5.1 *Baseline.*

We have provided the option to create and customize baseline models described in [6]. HolStep introduced a new data-set in higher-order logic. It features a few static data-sets that are trained on machine learning models fitted for the task of predicting whether a statement is useful in the proof of a given conjecture. The authors proposed specialized models focusing on two sub-tasks for this which train on unconditioned and conditioned classification of proof steps. In order to run these baseline models, you must

- Unconditioned classification of proof steps: determining how likely a given proof is to be useful for the proof it occurred in, based solely on the content of statement (i.e. by only providing the model with the step statement itself, absent any context).
- Conditioned classification of proof steps: determining how likely a given proof is to be useful for the proof it occurred in, with “conditioning” on the conjecture statement that the proof was aiming to attain, i.e. by providing the model with both the step statement and the conjecture statement).

3.5.2 *Architecture.*

The architecture for the baseline models cover a range of architecture features (from convolutional networks to recurrent networks), aiming at probing what characteristics of the data are the most helpful for usefulness classification. All of the models start with the same embedding layer, mapping tokens or characters in the statements to dense vectors in a low-dimensional space. For the scope of this project, we do not modify the topology of the two-tower neural architecture for ranking actions that is implementing in DeepHOL.

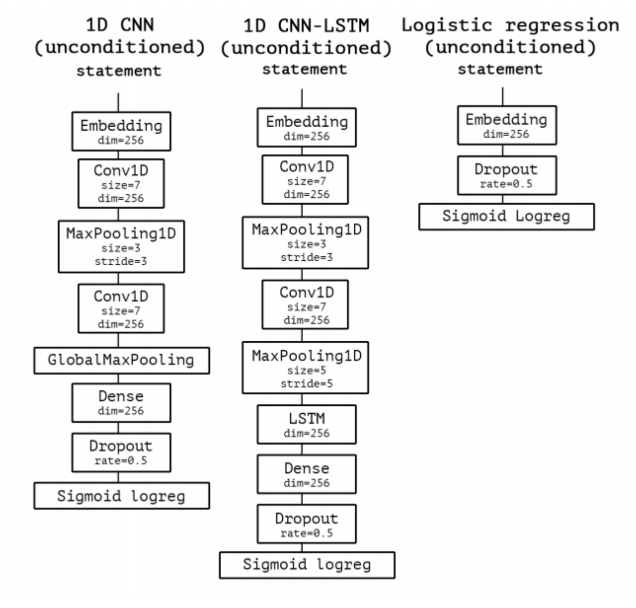

Figure 2: Unconditioned classification model architectures from [6]

The authors consider two possible encodings for presenting the input statements (proof steps and conjectures) to the model embedding layers:

- Character-level encoding of the statements, where each character is mapped to a 256-dimensional dense vector.
- Token-level encoding of the versions of the statements rendered with their own highlevel tokenization scheme.

3.5.3 Non-Neural Models. Standard machine learning techniques can be found using the `sklearn` option in the `--mode_type` flag. A naive bayes, Stochastic Gradient Descent regressor, and random forest classifier participate in a "bake-off". The feature space for these models are the same statement embeddings given to the neural networks. Feature engineering is left as a suggested improvement.

3.5.4 HyperParameter Tuning. An option for the keras models includes a grid search for hyperparameter optimization. This can be accessed with `run_type=search`. The default hyperparameter optimizations used in this grid search are:

```
learn_rate   = [0.001, 0.01, 0.1]
epsilon      = [None, 1e-2, 1e-3]
dropout_rate = [0.25, 0.5]
```

3.5.5 Suggested Improvements. The baseline models do simple pattern matching at the level of n-grams of characters or tokens. Their experiments found that their models do not appear to be able to leverage order in the input sequences, nor conditioning on the conjectures. Therefore, these models do not take advantage of order in the input sequences. This relationship could be better captured by Recursive Neural Tensor Networks or other graph-based recursive architectures. Wang et al. [7] found that their Chainer implementation of FormulaNet, a graph embedding based premise selection method for theorem proving, outperformed HOLStep baselines by 7-8 percent.

3.6 Provers

The task of proving a theorem is generally thought of as a tree search, where the root of the tree is a theorem to be proved, and each leaf node corresponds to an applied tactic. The Prover object is responsible for implementing the proof search. The default option in DeepHOL is a breadth-first search, and DeepHOL also implements a no-backtracking search for comparison. Other works have implemented different search algorithms for theorem proving such as Monte-Carlo search. [4]

Implementing a new prover in DeepHOL requires implementing a new subclass of the Prover class. All that is needed for this is to implement a custom constructor if needed, and the `prove_one` method, which takes a proof search tree, and a ProverTask as parameters, and return None when a proof is found, or an error message if the search fails. Example implementations can be found in prover.py.

3.7 Selection Strategies

The action generator is responsible for building the search tree by choosing tactics to apply, and selecting premises. This task is where the advantages of machine learning can be realized. The action generator allows us to intelligently choose tactics and theorems to apply to our goals, greatly narrowing our search space.

Tactic scoring and premise selection are handled by the abstract `Predictions` class located in predictions.py. New predictors can be implemented by subclassing the `Predictions` class, and implementing the following abstract methods. [2]

```
def _batch_goal_embedding(self, goals):
    """Computes embeddings from a
    list of goals."""

def _batch_thm_embedding(self, thms):
    """From a list of string theorems,
    computes and returns their
    embeddings."""

def proof_state_from_search(self, node):
    """Convert from
    proof_search_tree.ProofSearchNode to
    proof state."""

def proof_state_embedding(self, state):
    """From a proof state, computes and
    returns embeddings of each
    component."""

def proof_state_encoding(self, state_emb):
    """From an embedding of a proof state,
    computes and returns its encoding."""
```

Example implementations can be found at holparam_predictor.py, and a basic template can be found at
mock_predictions_lib.py.

Once these are implemented, the `get_predictor` function should be edited to return the new prover. To add a new configuration option, A new element should be added to the enum `ModelArchitecture` defined in deephol.proto, which should then be compiled as described in the *configuration* section of this tutorial.

4 JUPYTER ENVIRONMENT

We also provide a Jupyter notebook environment for rapid prototyping within DeepHOL. The source code is open-source and available at
github.com/aahadley/deepmath-jupyter/blob/master/HOLJup.ipynb.

The notebook consists of an interactive tutorial similar to the one found in this paper. After the tutorial, we provide explanations of the important functions of the prover. Each of these sections includes a code cell where the user can modify parts of the source code directly from the Jupyter notebook. This is accomplished by providing custom classes that import and return functions found in explicitly defined python files. If the appropriate user-defined function is not found, the class will fall back on default behavior defined in files that will not be overwritten by the Jupyter notebook.

5 CONCLUSION

We provide an overview and tutorial of the HOList environment. We provide instructions for modification within the DeepHOL prover, and present a Jupyter notebook environment allowing for rapid prototyping and ease of experimentation. In the near future, we hope to present an extension to the HOList environment allowing for interaction with the Coq proof assistant.

6 FUTURE WORK

HOList holds much potential for extension beyond what we have done here. The most immediately useful could be extending HOList to interact with other Proof Assistants such as LEAN, Coq, Isabelle, and HOL4. This will allow for training on more corpora and may be able to extract general features and discard features that may be particular to HOL-Light. GamePad [5] and CoqGym [8] both provide learning environments and an API for communicating with Coq. These may provide an easy starting point for setting up communication between Coq and DeepHOL.

Another area that shows potential for development is the application of new proving strategies. HOList employs a BFS for finding proofs in the proof search tree. Other works have seen success in applying AlphaZero style Monte Carlo search methods. [4] Beyond trivial implementations, we have not experimented with alternative selection strategies for premise selection and tactic scoring. We hope that with this tutorial, researchers in the future will be able to experiment and develop novel new approaches to these problems.

HOList created an environment and baselines for higher-order theorem proving. While the work done by the HOList team mainly focus on the tasks of premise selection and proof guidance, their open sourced framework can be tasked to other tasks relevant to theorem proving:

- Predicting which conjecture a particular intermediate statement originates from
- Predicting the name given to a statement
- Generating intermediate statements useful in the proof of a given conjecture
- Generating the conjecture that the current proof will lead to

REFERENCES

[1] [n.d.]. http://www.cs.ru.nl/~freek/100/

[2] [n.d.]. HOList. https://sites.google.com/view/holist/home

[3] Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, Christian Szegedy, and Stewart Wilcox. 2019. HOList: An Environment for Machine Learning of Higher-Order Theorem Proving (extended version). *CoRR* abs/1904.03241 (2019). arXiv:1904.03241 http://arxiv.org/abs/1904.03241

[4] Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. 2018. Learning to Prove with Tactics. *CoRR* abs/1804.00596 (2018). arXiv:1804.00596 http://arxiv.org/abs/1804.00596

[5] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. 2018. GamePad: A Learning Environment for Theorem Proving. *CoRR* abs/1806.00608 (2018). arXiv:1806.00608 http://arxiv.org/abs/1806.00608

[6] Cezary Kaliszyk, François Chollet, and Christian Szegedy. 2017. HolStep: A Machine Learning Dataset for Higher-order Logic Theorem Proving. *CoRR* abs/1703.00426 (2017). arXiv:1703.00426 http://arxiv.org/abs/1703.00426

[7] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. 2017. Premise Selection for Theorem Proving by Deep Graph Embedding. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 2786–2796. http://papers.nips.cc/paper/6871-premise-selection-for-theorem-proving-by-deep-graph-embedding.pdf

[8] Kaiyu Yang and Jia Deng. 2019. Learning to Prove Theorems via Interacting with Proof Assistants. *CoRR* abs/1905.09381 (2019). arXiv:1905.09381 http://arxiv.org/abs/1905.09381