Large Scale Deep Learning for Theorem Proving in HOList: First Results and Future Directions

Sarah Loos

HOList

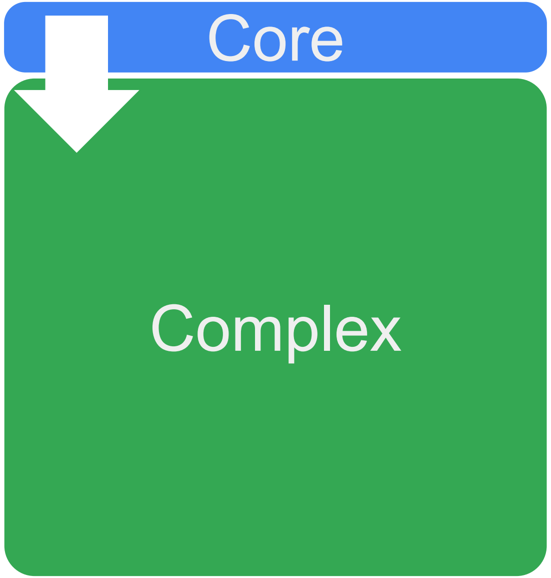
An Environment for Machine Learning of Higher-Order Theorem Proving

- HOList provides a simple API for ML researchers and theorem prover developers to experiment with using machine learning for mathematics.
- We use deep networks trained on an existing corpus of human proofs to guide the prover.
- We can improve our results by adding synthetic proofs (generated from supervised models and verified correct by the prover) to the training corpus.

Dataset Stats

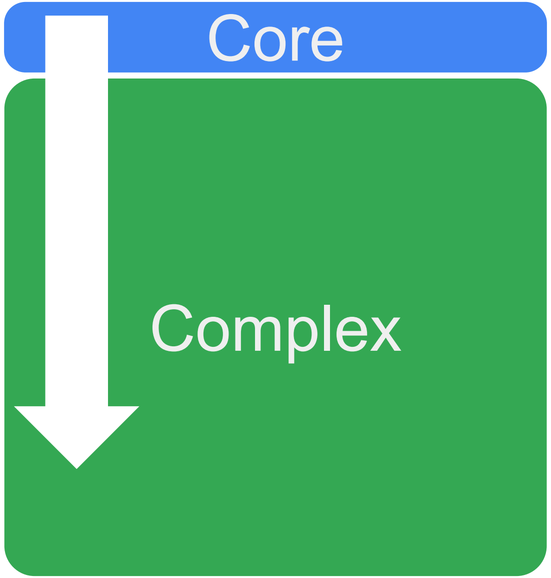
	Training 60%	Validation 20%	Testing 20%
Core	1.5K Theorems	500 Theorems	500 Theorems
Complex	10K Theorems	3.2K Theorems	3.2K Theorems

Dataset Stats



Training 60%	Validation 20%	Testing 20%
1.5K Theorems	500 Theorems	500 Theorems
10K Theorems	3.2K Theorems	3.2K Theorems

Dataset Stats



Training 60%	Validation 20%	Testing 20%
1.5K Theorems	500 Theorems	500 Theorems
10K Theorems	3.2K Theorems	3.2K Theorems

Dataset Stats

	Training 60%	Validation 20%	Testing 20%
Core	1.5K Theorems	500 Theorems	500 Theorems
Complex	10K Theorems	3.2K Theorems	3.2K Theorems

Dataset Stats

	Training 60%	Validation 20%	Testing 20%
Core	1.5K Theorems	500 Theorems	500 Theorems
Complex	10K Theorems	3.2K Theorems	3.2K Theorems
	375K Human Proof Steps	100K Human Proof Steps	100K Human Proof Steps

Dataset Stats

	Training 60%	Validation 20%	Testing 20%
Core	1.5K Theorems	500 Theorems	500 Theorems
Complex	10K Theorems **375K** Human Proof Steps	3.2K Theorems 100K Human Proof Steps	3.2K Theorems 100K Human Proof Steps
Flyspeck	**None**	10.5K Theorems	

Model Architecture

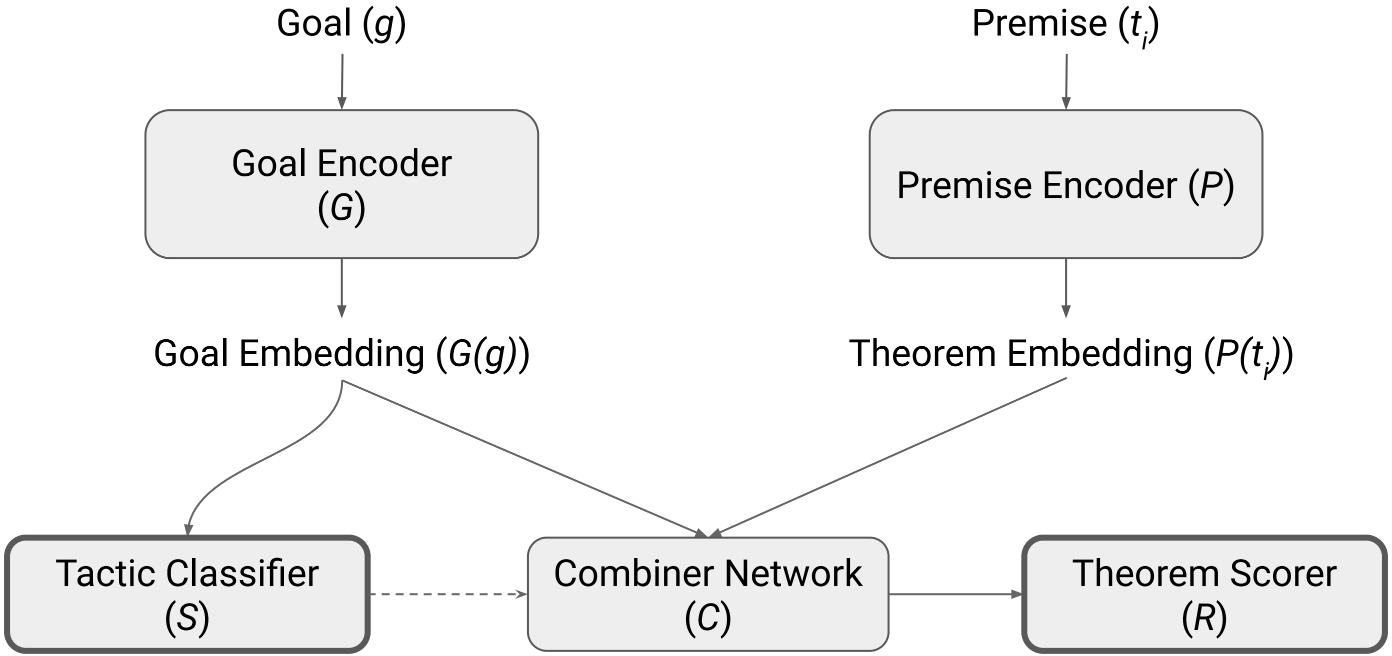


Figure courtesy of Viktor Toman

Results - Imitation Learning on Human Proofs

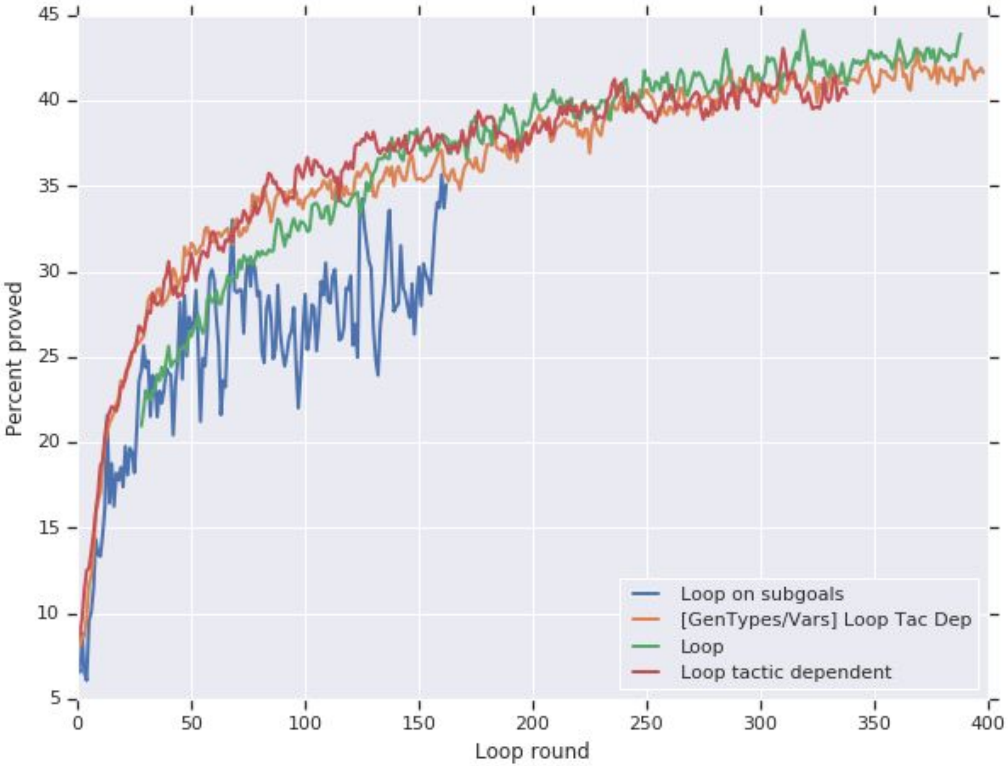
Model	Percent of Validation Theorems Closed
Baselines	
ASM_MESON_TAC	6.1%
ASM_MESON_TAC + WaveNet premise selection	9.2%
Imitation Learning	
WaveNet	24.0%

Reinforcement Loop: Setup

- In the reinforcement loop we train on a single GPU
- We simultaneously run search on multiple machines, each using the most recent checkpoint for proof search predictions.
- We run the neural prover in rounds, in each round trying to prove a random sample of theorems in the **training** set.
- Training examples are extracted from successful synthesized proofs and are mixed in with training examples from original human.
- Hard negatives: We omit arguments that do not change the outcome of the tactic application and store them as “hard negatives” for a specific goal to use during training.

Results - Reinforcement Loop

Training



Validation

	Percent Closed
Thin WaveNet Loop	36.30%
- Trained on loop output	36.80%
Tactic Dependent Loop	**38.90%**

Dataset Stats

	Training 60%	Validation 20%	Testing 20%
Core	1.5K Theorems	500 Theorems	500 Theorems
Complex	10K Theorems	3.2K Theorems	3.2K Theorems
	375K Human Proof Steps	100 Human Proof Steps	100 Human Proof Steps
Flyspeck	**None**	10.5K Theorems	

Dataset Stats

	Training 60%	Validation 20%	Testing 20%
Core	1.5K Theorems	500 Theorems	500 Theorems
Complex	10K Theorems **375K** Human Proof Steps **830K** Synthesized Proof Steps	3.2K Theorems 100 Human Proof Steps	3.2K Theorems 100 Human Proof Steps
Flyspeck	**None**	10.5K Theorems	

Results - Reinforcement Loop

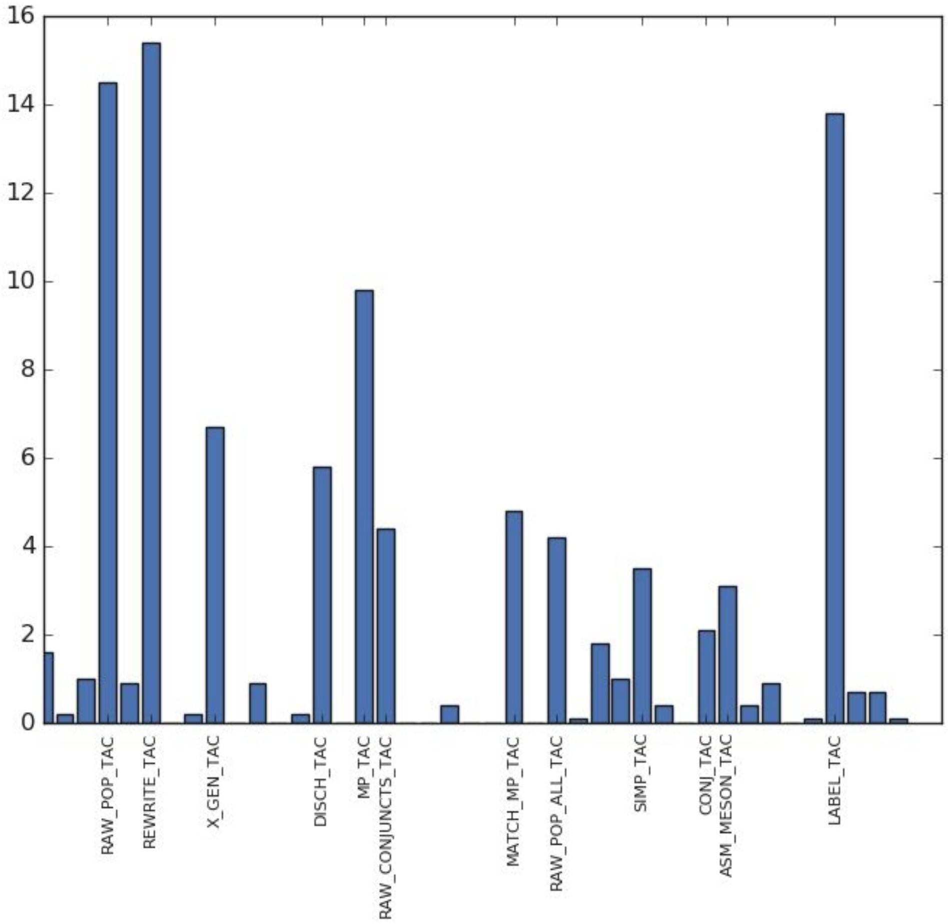
Model	Percent of Validation Theorems Closed
Baselines	
ASM_MESON_TAC	6.1%
ASM_MESON_TAC + WaveNet premise selection	9.2%
Imitation Learning	
WaveNet	24.0%
Imitation Learning + Reinforcement Loop	
WaveNet	36.3%
- trained alongside output	36.8%
Tactic Dependent	**38.9%**

Results - Reinforcement Loop

Model	Percent of Validation Theorems Closed
Baselines	
ASM_MESON_TAC	6.1%
ASM_MESON_TAC + WaveNet premise selection	9.2%
Imitation Learning	
WaveNet	24.0%
Imitation Learning + Reinforcement Loop	
WaveNet	36.3%
- trained alongside output	36.8%
Tactic Dependent	**38.9%**

Flyspeck:
On a sample of 2000 proofs from the flyspeck dataset
37.6%

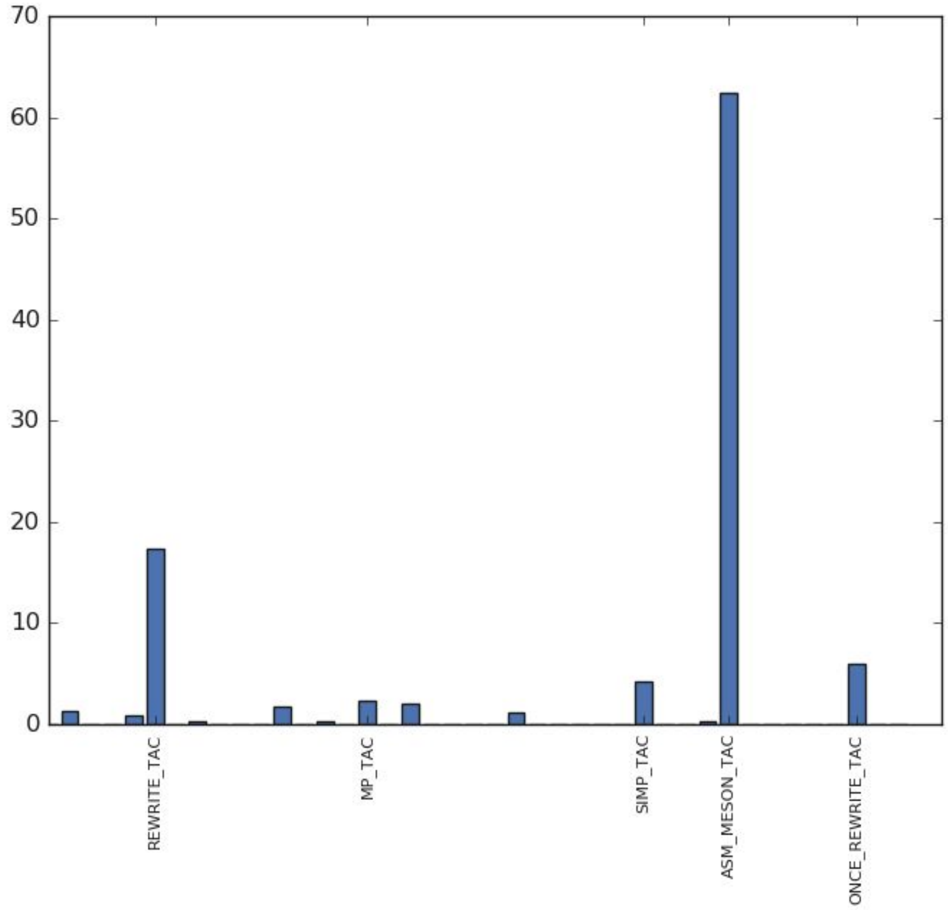
Tactics Distribution - Human Proofs



Most commonly used human tactics:

- REWRITE_TAC
- RAW_POP_TAC
- LABEL_TAC
- MP_TAC
- X_GEN_TAC

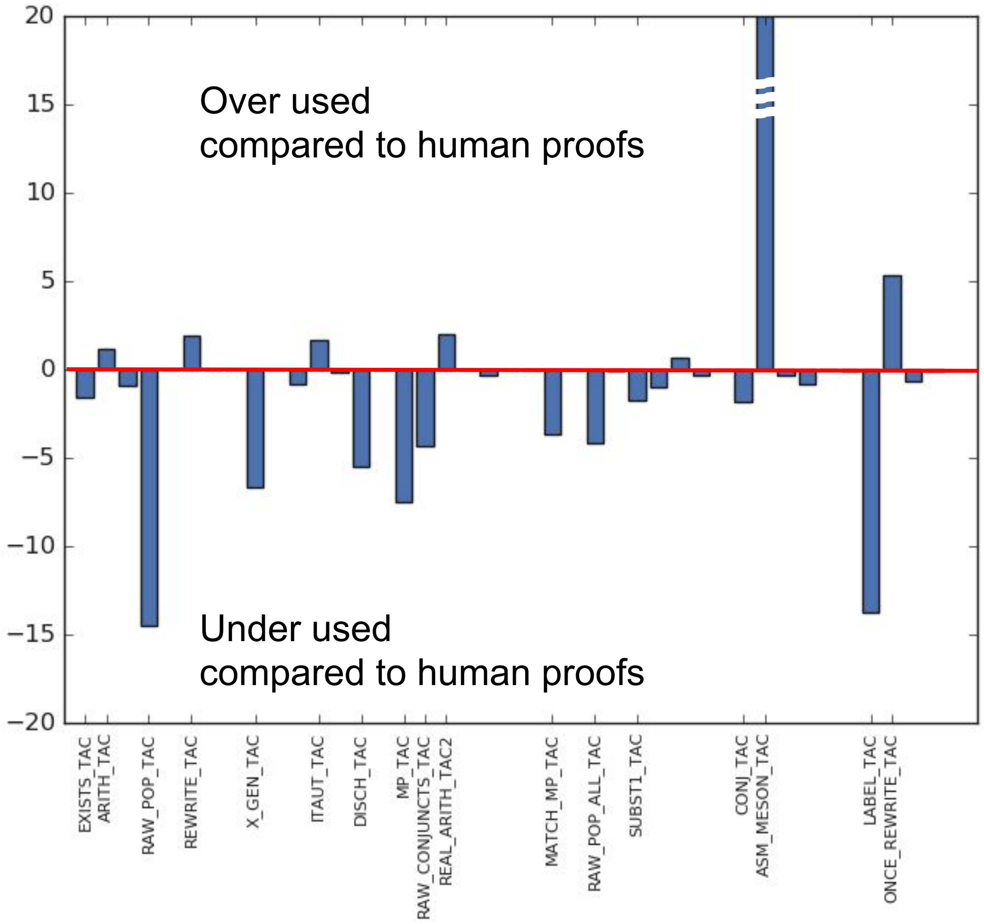
Tactics Distribution - Reinforcement Loop



Tactics used in Reinforcement Loop:

- ASM_MESON_TAC
- REWRITE_TAC
- ONCE_REWRITE_TAC
- MP_TAC
- SIMP_TAC

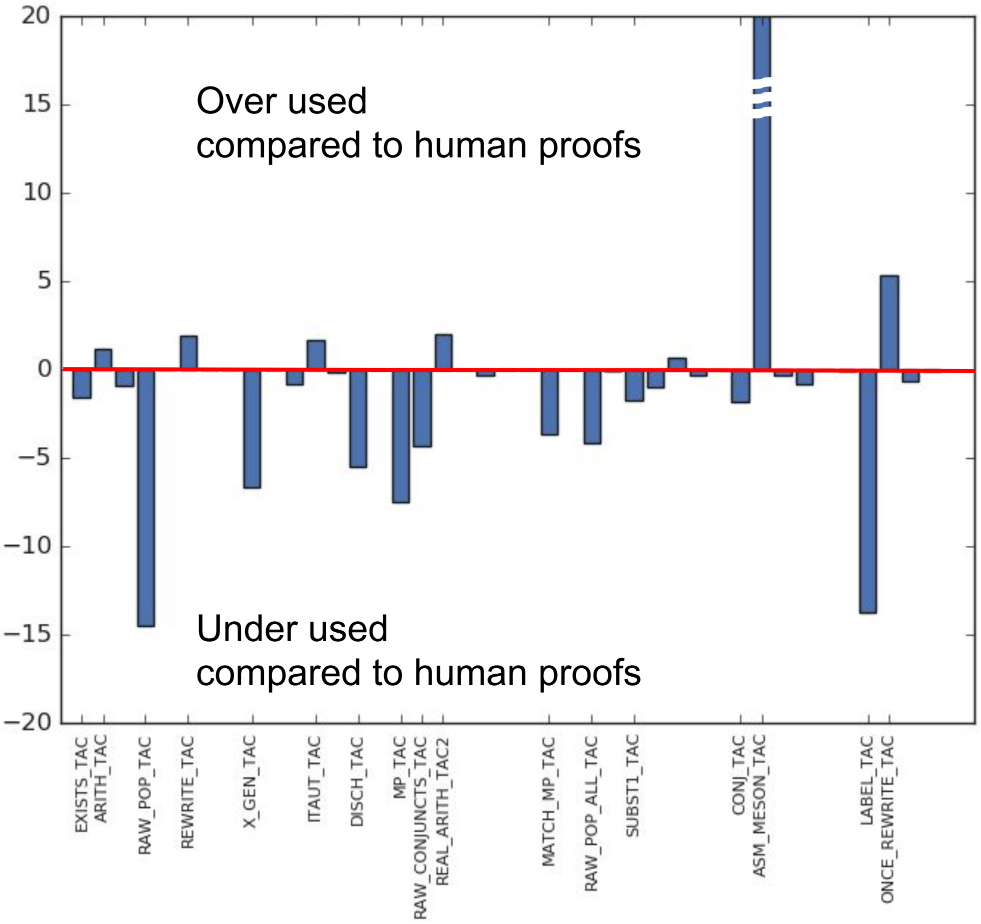
Tactics Comparison



Most increased:
- ASM_MESON_TAC
- ONCE_REWRITE_TAC

Most decreased:
- LABEL_TAC
- RAW_POP_TAC
- MP_TAC
- X_GEN_TAC

Tactics Comparison



Most increased:
- ASM_MESON_TAC
- ONCE_REWRITE_TAC

Most decreased:
- LABEL_TAC
- RAW_POP_TAC
- MP_TAC
- X_GEN_TAC

Soundness is Critical

ITPs motivated by concerns around correctness of natural mathematics.

- HOL Light relies on only ~400 trusted lines of code.

You should not need to trust more than that:

- Environment optimizations: startup cheats-ins and proof search code are now in the critical core (!) -- we must have a proof checker.
- **Reinforcement learning reinforces soundness problems.**

Proof Checker

We provide a proof checker that compiles proof logs into OCaml code

- Human-readable format
- Can be checked with HOL Light’s core

To be sure that the proofs work, the proof checker replaces HOL Light’s built-in proofs by the imported synthetic proofs.

- Same soundness guarantees as HOL Light.

Proof Checker - Example

Goal: `|- !x y. exp (x - y) = exp x / exp y`

```
Parse_tactic.parse "ONCE_REWRITE_TAC [ THM 1821089176251131959 ]" THEN
Parse_tactic.parse "ONCE_REWRITE_TAC [ THM 4045159953109002127 ]" THEN
Parse_tactic.parse "REWRITE_TAC [ ]" THEN
Parse_tactic.parse "SIMP_TAC [ THM 3715151876396972661 ;
                              THM 1821089176251131959 ;
                              THM 2738361808698163958 ]" THEN
Parse_tactic.parse "ASM_MESON_TAC [ THM 4334187771985600363 ;
                                   THM 1672658611913439754 ;
                                   THM 4290630536384220426 ;
                                   THM 3714350038189073359 ]"
```

Proof Checker - Example

Goal: `|- !x y. exp (x - y) = exp x / exp y`

```
ONCE_REWRITE_TAC [ FORALL_UNPAIR_THM ] THEN
ONCE_REWRITE_TAC [ FORALL_PAIR_THM ] THEN
REWRITE_TAC [ ] THEN
SIMP_TAC [ MESON[] `(!t. p t) <=> ~(?t. ~p t)` ;
            FORALL_UNPAIR_THM ;
            real_div ] THEN
ASM_MESON_TAC [ REAL_EXP_NEG          (* |- !x. exp(--x) = inv(exp(x)) *) ;
                REAL_POLY_CLAUSES     (* includes induction on exp *) ;
                REAL_EXP_ADD_MUL      (* |- !x y. exp(x + y) * exp(--x) = exp(y)*) ;
                REAL_EQ_SUB_LADD      (* |- x y z. (x = y - z) <=> (x + z = y)*) ]
```

Hard Negative Mining

- During training, we can simultaneously mine hard negatives by ranking all theorems and adding extra training on negative examples ranked just above positives.
- This is an early result, but it seems to help a lot for imitation learning.
- Next step: Try it in the reinforcement loop.

Results - Hard Negative Mining

Model	Percent of Validation Theorems Closed
Baselines	
ASM_MESON_TAC	6.1%
ASM_MESON_TAC + WaveNet premise selection	9.2%
Imitation Learning	
WaveNet	24.0%
With Hard Negative Mining	37.2%
Imitation Learning + Reinforcement Loop	
WaveNet	36.3%
- trained alongside output	36.8%
Tactic Dependent	38.9%

Challenges: Learning for Theorem Proving

- Infinite, very heterogeneous action space
- Extremely sparse reward
- Unbounded, growing knowledge base
- Infeasibility of self-play/self-play is not obviously employed (the way it is known from chess or go)
- Slow evaluation

Discussion

- RL Loop - Zero shot learning.
- Suggestions from other work (e.g. imitation learning, from AlphaStar).
- Opportunities for the community.
- http://deephol.org (Code is on GitHub. Training data, checkpoints, docker images also being made available.)
- Arxiv preprint: https://arxiv.org/abs/1904.03241, **"HOList: An Environment for Machine Learning of Higher-Order Theorem Proving"**