Policy Network, Exploration and Enhancement for Proving Large Theories

CS285 Final Project

Noah Gundotra[1], Yulun R. Wu[1], Daniel Zeng[1]

UC Berkeley[1]

Abstract

Automated theorem proving has been notoriously difficult for reinforcement learning to work with because of its large state space and compute-intensive simulations. In this paper, we attempt to overcome these challenges by combining supervised learning methods with exploration-based reinforcement learning methods. In particular, we build from earlier work that trains a model to predict combinations of tactics to apply on proofs in the Theorem Assistant Coq. We use a model that is trained on human proofs, which allows the deep learning model to learn useful embeddings across a collection of 71k proofs. Training a model this way creates useful embeddings, and allows fine-tuning of the decoding mechanisms with reinforcement learning. In order to increase the number of proofs the deep learning model is able to predict, we collect asynchronous trajectories and backpropagate with policy gradients, augmented by a RND-based exploration bonus. We conduct an ablation study over 3 different methods with and without using the RND-based exploration bonuses. These methods include: vanilla sampling, DFS sampling, vanilla sampling with a frozen encoder. Our experiments show that these methods were not sufficient to improve performance of the deep-learning model on the proofs we experimented on.

Keywords: Deep Reinforcement Learning, Interactive Theorem Proving, Policy Gradients, Beam Search, Exploration

1 Introduction

Science needs automated verification and cross-referencing. Today's experts are often so specialized within their own field, that experts are unable to communicate across academic domains. This is a modern phenomenon, and is only increasing in scale. It would be an amazing phenomenon to have software-based verification and querying of today's scientific breakthroughs. One way this could be accomplished is

through the translation of scientific statements into formally-verifiable statements. A place to look is in the formal verification of mathematical statements.

Development of mathematical theorem assistants has a four decade long history. For example, notable theorem assistants include Coq, Lean, HOL, and Isabelle. These theorem assistants avoid the NP-Complete problem of autonomous theorem proving by requiring input from the user and providing detailed feedback. These systems are collectively referred to as interactive theorem provers (ITPs). However the large variation in systems and their mathematical foundations show that there is no currently accepted consensus on how to best balance proof readability, verification performance, and proof structuring.

Recent successes of reinforcement learning in very complex domains, such as playing Go (Silver et al., 2016), solving a Rubik's cube using a robot hand (OpenAI et al., 2019), and playing in imperfect information games (Brown et al., 2020), would lead one to believe we are close to automated theorem proving as well. However, we appear to be far from there. The large hurdles preventing progress in deep-learning based theorem proving lies with the heavy CPU costs and low economic incentive of searching for proofs of known mathematical statements. We attempt to overcome the CPU cost of simulation by pretraining the model on human proofs with supervised learning techniques, and supersede state of the art performance by fine-tuning the model with exploration-augmented policy gradients.

2 Related Work

CoqGym is the most relevant related work that tackles theorem proving through automation interactions with the interactive proof assistant Coq (Yang and Deng, 2019). They have two major introductions: the collection of 71 thousand proofs across various scientific domains, and the introduction of ASTactic, which we will discuss in our methods section. In CoqGym, the authors train their ASTactic model to predict proofs given a theorem statement by training their model over their 71 thousand proof data set. It is worth noting that the authors only use supervised learning techniques to train their model. They report an improvement over other popular community programs including *hammer*, and even being able to prove theorems where hammer cannot. Hammer is the largest program written in Coq that combines years of human expertise and knowledge into heuristics that attempt to prove theorems. Their results also show that they are able to prove even more theorems by combining hammer and ASTactic together. However, their results also reveal room for improvement, since ASTactic can only prove 12.2% of the theorems in the dataset.

Another related work, HOList tackles theorem proving through interacting with the HOLight interactive proof assistant (Bansal et al., 2019b). HOList introduces a reinforcement learning environment that allows interaction with the HOLight ITP in Python, and deep learning prover architecture. Their dataset in HOLight consists

of 29 thousand proofs that are related to the Kepler Conjecture, which has taken 20 years to prove and formalize. Their neural prover differs from ASTactic because they predict one tactic at a time, and have a network devoted to ranking which tokens should be given to each tactic as arguments. It is worth noting that they punish unused tactic arguments in their training, and they spend extra computation time to greedily prune their argument lists to determine which arguments were necessary. Using this method, they are able to prove at best 39% of their flyspeck dataset. Subsequent work from the same authors in ZeroExplore combines human proofs to perform imitation learning and exploration rewards such as term inverse document frequency to reward their neural prover. The best ZeroExplore method is able to prove 60% of their flyspeck dataset (Bansal et al., 2019a).

3 Contributions

We draw upon earlier work for our deep-learning model architecture and action-selection mechanisms. In particular, we use the ASTactic model pioneered [CoqGym citation] pretrained across their 71 thousand human proofs. This was chosen because of the large range for improvement and the fact that their publication-ready code was publicly available. In our paper we demonstrate the results of fine tuning the pre-trained ASTactic model with three policy gradient based methods:

1. Vanilla — No Exploration
2. Vanilla — Exploration
3. Vanilla + Frozen Encoder — No Exploration
4. Vanilla + Frozen Encoder — Exploration
5. DFS — No Exploration
6. DFS — Exploration
7. DFS + Frozen Encoder — No Exploration
8. DFS + Frozen Encoder — Exploration

These methods are explained below and results are presented in our experiments section.

4 Methods

4.1 ASTactic

Our approach uses the ASTactic neural proof prediction model introduced in CoqGym. The pre-trained model is acquired by ASTactic model, which trains a recurrent network for tactic selection with supervised learning. The model encodes

the state of the current environment with a Tree LSTM, and builds an AST with GRUs to represent the application of several related tactics. When building the AST, the GRU decoder expands the non-terminal nodes of the tree into another branch, a terminal tactic, or a token of a terminal tactic. The unique contribution of ASTactic from the CoqGym paper is the ability to produce a series of related tactics at once. This is why the output of ASTactic is referred to as a program.

During training, the model does not interact with CoqGym. Instead, the model learns to imitate human programs by treating them as ground truth prediction. To accommodate for the fact that ASTactic may not output comparable programs to the ground truth, ASTactic is trained with teacher forcing. Teacher forcing makes ASTactic predict programs from a partially expanded ground-truth proof at each time step instead of predicting programs from scratch, so losses can be measured. To evaluate the model, the model's predicted programs are initialized with a "tactic template" before being sent to the Coq proof system. Tactic templates can be a series of other built-in Coq programs such as 'auto', 'intuition', or 'hammer'.

CoqGym uses ML to encode states and learns tactic selection using supervised learning, and this is imitation learning, and does not use reinforcement learning loops to interact and learn with the environment. We expand on this innovation using policy gradients and exploration, which is further explored in the lower sections.

4.2 Distributed Policy Gradients

4.2.1 Policy Gradients

We take the ASTactic as our baseline model and further train it with online reinforcement learning. Through interacting with the real environment, the proving agent should be able to receive additional feedback and yield results that are more robust than purely imitating human proofs. For a proving trajectory, we assign a living penalty of -0.1 per step, reward of 20 for successful proof and penalty of -1.5 for failure. The objective function and the policy gradient are given by

$$J(\theta) = \mathrm{E}_{\tau \in p_\theta(\tau)}[r(\tau)] \tag{1}$$

$$\nabla_\theta J(\theta) = \mathrm{E}_{\tau \in p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\mathbf{a} \mid \mathbf{s}) \cdot r(\tau)] \tag{2}$$

A supplementary critic network would be a great addition to policy estimation for the purpose of variance reduction, but it is not a focus of this paper and will be left to future work.

4.2.2 Parallel Sampling

The need for distributed policy gradients arises from the need to parallelize the proof trajectory collection process. This is because interacting with the environment, which is the Coq proof assistant in our case, incurs high I/O latency due

to the need to communicate with the Coq API via sockets. We distribute trajectory collection over many workers, allowing parallelization of this process. However, because each worker collects its own trajectories and as a result different update gradients associated with the trajectories, policy gradient is required to aggregate these gradients, which is done in the coordinator thread. The coordinator thread then uses the aggregated gradient and finally updates the model, which is set back to the workers for additional proof trajectory collection.

4.2.3 Tactic Decoder Fine-tuning (Frozen Encoder)

Since reinforcement learning is used to improve off the base model trained in the imitation learning setting, we experiment with two scenarios. The first scenario, as defined above, is in which we train policy gradients in the vanilla way. In the second scenario, we freeze the encoder used to generate tactic representations, and then fine-tune the tactic decoder to further optimize tactic selection.

4.3 Exploration: Random Network Distillation

For exploration, we use the technique of Random Network Distillation (Huang et al., 2018). In essence, we want to reward the agent for performing actions that lead to novel states, and thus expand its own set of visited states. To do exploration, Random Network Distillation creates two networks, both of which are initialized randomly with different initialization methods at the start of the algorithm. One network is fixed $f_\theta(s')$, and the other network learns to approximate the fixed network through $\hat{f}_\phi(s')$.

As the model starts training, the reward bonus is augmented to where the error of the learning model is high, and this represents states in which the agent has visited less. As the agent visits certain states more, network distillation would decrease the error of these models.

We initialize one Random Distillation network with a Xavier Uniform distribution and another with a Normal distribution in order to compute distance in state distributions. The input terms are passed into a frozen term encoder, as done previously, and then fed into a final layer which maps each embedding in the sequence to a scalar output. The distance between the output of random and trained network is then given as the bonus exploration reward. The objective of the trained network is to minimize this distance on $\mathcal{D}$:

$$\phi^* = \underset{\phi}{\text{argmin}}\ \mathrm{E}_{s,a,s' \sim \mathcal{D}}[||\hat{f}_\phi(s') - f_\theta(s')||] \tag{3}$$

4.4 Enhancement: Controlled Beam Distillation

One innovation of our approach is that we further refine the policy under a more controlled setting. We drew our inspiration on one work which learns beam search policies using imitation learning (Negrinho et al., 2018). During test time, our model carries out a depth-first search ordered by the likelihood of top tactics recommended by the policy network, until the proof is complete or the maximum number of permitted tries is reached. We want to extend this technique to training time so that we could distill the top tactics with dissatisfactory performance. The training instances are collected with the approach given below.

Algorithm 1 Controlled Beam Pseudo Sampling

Require: g, env, $pnet$

$tacs, \pi s \leftarrow pnet(g)$ ▷ The viable tactics and their likelihoods
$stack \leftarrow \{\text{top n } tacs \text{ with } \pi s\}$
$traj, samples \leftarrow \{\}, \{\}$ ▷ Current trajectory and accumulated samples
while not (*stack* empty **or** proof succeeded **or** max steps reached) **do**
    $tac, \pi \leftarrow stack.pop()$
    add $\{g, tac, \pi, ?\}$ to $traj$ ▷ Last place saved for reward
    $g \leftarrow env.step(g, tac)$
    if proof ends **then**
        assign reward to $traj$
        add $traj$ to $samples$
        $traj.pop()$
    else
        $tacs, \pi s \leftarrow pnet(g)$
        $stack \leftarrow \{\text{top n } tacs \text{ with } \pi s\}$
return $samples$

Theoretically, if we desire to conduct this approach in a more rigorous manner, we ought to make the stopping time independent of the proof result of each trajectory. The expected gradient can then be approximated by

$$\nabla_\theta J(\theta) = \mathrm{E}_{\tau \in p_{\theta'}(\tau)}[\frac{p_\theta(\tau)}{p_{\theta'}(\tau)} \nabla_\theta \log \pi_\theta(\mathbf{a} \mid \mathbf{s}) \cdot r(\tau)] \tag{4}$$

$$= \sum_{i=1}^{n} (\prod_{t=1}^{T_i} \frac{\pi_\theta(a_t \mid s_t)}{N(s_t, a_t)}) (\sum_{t=1}^{T_i} \nabla_\theta \log \pi_\theta(a_t \mid s_t)) (\sum_{t=1}^{T_i} r(s_t, a_t)) \tag{5}$$

where $N(s_t, a_t)$ denotes the totally number of times tuple (s_t, a_t) is visited across all trajectories, and the equality is strict since the transition dynamic is fixed in the situation we are dealing with. Note that this estimator is different with the importance sampling estimator in that the sample paths are in fact deterministically

drawn, similar to exploration with Monte Carlo Tree Search. Intuitively, this is also comparable to approximating gradients in a deterministic manner under continuous settings, such as drawing a uniform grid which would yield a trivial error bound $L_1(\nabla_\theta \hat{J}(\theta), \nabla_\theta J(\theta)) \leq \delta L_{f_\theta} \cdot p(E_{f_\theta})$ when f_θ is uniformly continuous, where $f_\theta(\tau) = p_\theta(\tau)\nabla_\theta \log \pi_\theta(\mathbf{a} \mid \mathbf{s}) \cdot r(\tau)$, L_{f_θ} is its Lipschitz bound and E_{f_θ} is its support. With our discrete theorem proving environment, the gradient evaluation would be exact if we traverse through all possible tactics. Yet this is obviously unrealistic, and does not align with our purpose of distilling either since we do not care about the tactics that lie on the tail of the distribution.

For this reason, we draw training instances with likelihood-ordered traversing and terminate the process as soon as a trajectory successfully proves the theorem or the maximum number of tries has been reached. The importance weights are also discarded so that the process punishes the repeatedly failing tactics more heavily. In this fashion, the gradient is also naturally linked to proving time cost, since the initial steps of a successful trajectory would receive less reward if it failed frequently before the theorem is eventually proven.

5 Experiment

For our experiments, we chose to train on 1 closed environment and 1 open environment to operate on. The closed environment, checker, contains 4 proofs about the checkers game. The open environment, group-theory, contains 57 proofs about basic group theory identities, however we train on only 9 of these proofs, which is why we consider it an "open" environment.

For the checker environment we train each method for 50 epochs. For the group-theory proofs we train on 20 epochs with a timeout of 30 seconds. These are comparable amounts of data because the checker environment has 4 proofs, and the group-theory environment has 8 proofs. For the policy gradient methods, we collect 4 sets of trajectories over the environment per epoch because we can collect different trajectories asynchronously. However for the DFS methods, we only collect 1 set of trajectories over the environment per epoch because they are all the same until we perform an update. Our results are reported in Table 1.

Vanilla sampling methods underperformed on the checker environment and only beat baseline on group-theory with the frozen encoder and RND exploration bonus. This makes sense because these methods are driven by random sampling. So we can only improve upon the pretrained model by guiding it to states it has never seen before and if those states lead to successful proofs. However on a small and straightforward proofs like in checker, there are no intermediary states to success, so the exploration bonus gradually decreases to 0. Conversely, this explains the success of the RND and exploration bonus variant succeeding in the context-rich state space of the group-theory proofs. A graph of the exploration bonus over the number of epochs are show in Figure 1.

Environment	Sampling	Not Frozen	Frozen	RND	Frozen + RND	Best
Checker	Vanilla	0/4	0/4	0/4	0/4	0/4
	DFS	0/4	0/4	1/3	1/3	1/3
	None	-	-	-	-	**1/3**
Group Theory	Vanilla	0/9	0/9	0/9	2/7	**2/7**
	DFS	2/7	2/7	2/7	1/8	**2/7**
	None	-	-	-	-	1/8

Table 1: Comparing performance and ablation study of fine-tuning algorithms on the number of proofs we were able to prove. Numbers listed are given as number of theorems proved / number of proofs failed. The None sampling method refers to the baseline pretrained model.

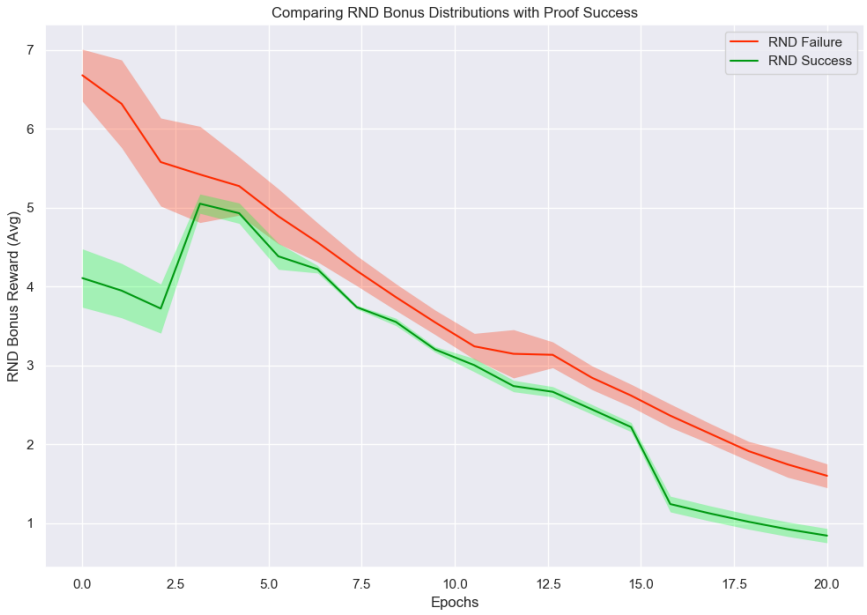

Figure 1: Plot of RND Bonus Distribution against Epochs

DFS sampling methods almost entirely failed on the checker environment but outperformed on the group-theory environment with the exception of the frozen encoder and RND exploration variant. This is a little harder to explain on checker, because we expected DFS algorithm to outperform the pretrained model because depth first search should be able to predict the same successful tactics as the pretrained model.

6 Future Work

In trying to improve on ASTactic on CoqGym, we found some interesting flaws that caused our models to be crippled. The issue that causes this is the inclusion of several heuristics in the context free grammar that ASTactic searches over. The ASTactic model learns to only apply high-level tactics over and over again-namely 'auto', 'intuition', and 'easy'. A related issue is that the ASTactic model will consistently output the same tactic during beam search applications even if it has already tried and failed to apply that tactic. Future work could remove these tactics from the training process of ASTactic and compare model performance, however this would also require creating new synthetic proof datasets from human proofs.

The simplest new work to be done would be increasing the depth and size of the encoder and decoder used in ASTactic and retraining on the 71k proofs. This is also expensive, but could result in a large improvement in the performance of the algorithm, considering the model's weights only amount to 28 Mb of memory.

7 Conclusion

We applied several reinforcement learning algorithms on the CoqGym environment with a pretrained neural prover. During this process, we learned about the limits of current neural provers on ITPs, and demonstrated that these reinforcement learning methods are not enough to improve upon existing work.

References

Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, and Christian Szegedy. Learning to reason in large theories without imitation. *CoRR*, abs/1905.10501, 2019a. URL `http://arxiv.org/abs/1905.10501`.

Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, Christian Szegedy, and Stewart Wilcox. Holist: An environment for machine learning of higher-order theorem proving (extended version). *CoRR*, abs/1904.03241, 2019b. URL `http://arxiv.org/abs/1904.03241`.

Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games, 2020.

Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. *CoRR*, abs/1810.12894, 2018. URL `http://arxiv.org/abs/1810.12894`.

Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. *CoRR*, abs/1806.00608, 2018. URL `http://arxiv.org/abs/1806.00608`.

Renato Negrinho, Matthew Gormley, and Geoffrey J Gordon. Learning beam search policies via imitation learning. *Advances in Neural Information Processing Systems*, 31:10652–10661, 2018.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019. URL `http://arxiv.org/abs/1910.07113`.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. *CoRR*, abs/1905.09381, 2019. URL `http://arxiv.org/abs/1905.09381`.