Learning theorem proving through self-play

Stanisław Purgał

University of Innsbruck, Innsbruck, Tirol, Austria
stanislaw.purgal@uibk.ac.at

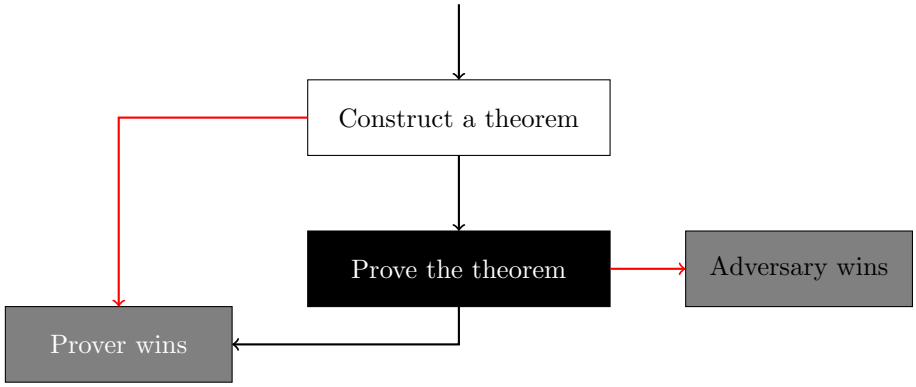
1 Introduction

This work attempts to apply the AlphaZero algorithm [4] to theorem proving. Following the philosophy of learning without using any human–generated datasets we attempt to learn to prove theorems without using any database of proofs or theorems. This is different from other attempts at ML-guided theorem proving in [2], [3], [1].

The only input we expect before starting training is the set of axioms we can use in our proofs — no theorems or conjectures.

2 The theorem-construction game

In the game we are using to learn theorem proving, one player constructs a provable theorem and the other player tries to prove it:



The goal of the adversary is to construct such a theorem, that the prover will fail to prove it. Because of the way the construction works, this theorem will have to be provable.

In the game we use prolog-like *terms*, where a term can be either a *variable*, or a pair of an atom and a list of subterms. In the examples we use the convention of marking variables with capital letters, and denoting compound terms and an atom name followed by a list of subterms in brackets (skipped when the list is empty).
Eg. `node(A, leaf)`.

The construction game is defined for a given set on *inference rules*. An inference rule is a pair of a term and a list of terms, that can share variables.
Eg. `tree(node(A, B))` ← `tree(A)`, `tree(B)`.

A state here is a pair consisting of a list of terms that need to be proven and an information about which player is now in control. During its move a player can choose one of the given inference rules, and apply it to the first term of the list. The left side of the rule is then unified

with that term. If the unification fails, the player making the move loses. If it succeeds, the term is removed from the list, and the right side of the rule (after unification) is added.

The first player (called *adversary*) starts the game with a list consisting of a single variable term. It then proceeds to "prove" it using the inference rules. As it is a variable, to begin with any inference rule can be applied. When the list is empty (meaning that the theorem was proven), the variable we started with will be unified with some theorem. This theorem is then given to the other player, after replacing every remaining variable with a fresh ground atom.

The second player then tries to prove the theorem, winning when the list is empty.

To ensure termination of the game, during every move there is a small chance that the player making the move will immediately lose, so that every game will end with probability 1.

3 Monte Carlo tree search modification

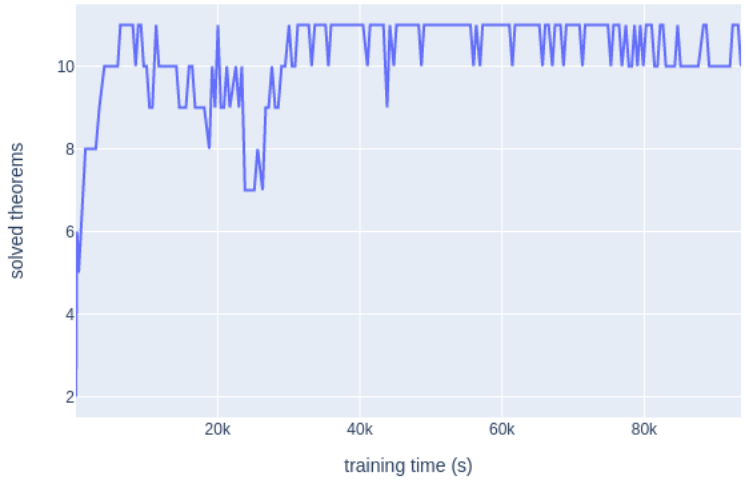
The AlphaZero [4] algorithm utilizes the Monte Carlo Tree Search (MCTS) to estimate state values and policies. As it is used there it works well, when getting a sure value of a game state is almost impossible. However, when players don't take turns, and instead can make several moves in a row, it's possible to find a path to a winning state, and prove with certainty the value of state without searching infeasibly large state space.

To allow propagation of sure state values in our implementation of MCTS we keep track of upper and lower bound for every state. In a non-final game state these are simply (1) and (-1) (as the reward is always somewhere between -1 and 1), but in a final state they are both equal to the outcome of the game. These bounds are then propagated up the tree, in accordance with state ownership (with which player is making a move in which state). This assures that if the tree search finds a certain way for one player to win in state s, the value of this state will become exactly 1.

It is worth pointing out that finding a winning path in the MCTS doesn't necessarily mean further search is pointless. Eg., the player constructing the theorem can avoid building theorems, for which the MCTS already found a proof.

4 Preliminary investigation

When training our model on a toy problem (involving reversing a list) we observed that the performance does improve in time, although it does not achieve a stable high result. Although we do not use any set of theorems during training, we do require it to measure the performance.



For estimating value and policy we currently use a variant of a Graph Attention Network [5], but we plan on experimenting with different architectures, as well as different axiom sets and hyperparameters.

References

[1] Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, and Christian Szegedy. Learning to reason in large theories without imitation. *ArXiv*, abs/1905.10501, 2019.

[2] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olsák. Reinforcement learning of theorem proving. In *NeurIPS*, 2018.

[3] Michael Rawson and Giles Reger. A neurally-guided, parallel theorem prover. In *FroCos*, 2019.

[4] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017.

[5] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2017.