Dreaming to Prove

Kristóf Szabó[1,2] and Zsolt Zombori[1,2]

[1] Alfréd Rényi Institute of Mathematics, Budapest
[2] Eötvös Loránd University, Budapest

Introduction *World models* represent the basic mechanisms of a system and can provide predictions about how transformations (actions) affect the state of the system. Such models have recently gained attention in Reinforcement Learning (RL) and in several domains model based learning systems performed similarly or better than highly tuned model free variants [1, 8, 12]. World models can increase sample efficiency since trajectories can be generated without interacting with the environment, and they can aid exploration by yielding a semantically meaningful latent structure that allows for identifying promising directions.

Our project raises the question whether such world models are achievable for theorem proving, i.e. whether state-of-the-art machine learning toolset is capable of capturing the underlying dynamics of an Automated Theorem Proving (ATP) system. A world model for an ATP system should know what moves are valid and when the proof search failed or succeeded.

An ATP system can often be framed as an RL environment: In each state of the prover, it has to select from a set of valid inferences (actions) that result in a new state, and a proving agent aims to select inferences that maximise its chance of finishing the proof. We select the leanCoP [6] connection tableau calculus for which we try to build a world model. leanCoP has a clean and compact notion of a *state* to which a limited number of inferences (*actions*) can be applied, making it a comfortable RL environment. Our two primary questions are: 1) what is length barrier within which the model can generate valid inference sequences and 2) to what extent the model can predict if an inference leads us closer to the end of the proof.

Most related to our work is [4], which train a latent embedding model, a state transition model in latent space, and a model that predicts the applicability of rewrite steps solely from the latent embedding. The trained models can predict rewrite step validity significantly better than simple baselines even after 10 rewrites in latent space. However, evaluation is only performed on valid rewrites, so we do not know how far the model can chain its own predictions. Furthermore, little is known about how much the state changes during the rewrites, while the inferences in leanCoP have a clear sense of directionality (there is no returning to the same state).

We turn leanCoP into an RL environment, as done in [2, 11, 10] and adapt the DreamerV2 [1] model-based RL algorithm which shows impressive results in modeling ATARI video games.

Dreamer Architecture The Dreamer system uses several neural network components to map observed states into a latent representation and then to perform actions in latent space. Given state space S and action space A, we train an *encoder* model $E : S \to \mathbb{R}^n$ which creates a latent vector. We also train a *decoder* $D : \mathbb{R}^n \to S$ which reconstructs the state from latent code. A *reward prediction model* $R : \mathbb{R}^n \to \mathbb{R}$ estimates the reward from latent code associated with the given state. Finally, a *transition* model $T : (\mathbb{R}^n, A, X) \to (\mathbb{R}^n, X)$ predicts the effect of an action on the latent code. T is implemented as a recurrent state-space model (RSSM), equipped with its own internal state $x \in X$, that is updated after each transition. All components are trained jointly, on sequences of (s_i, a_i, r_i, s_{i+1}) state, action, reward, successor tuples, using the following three objectives: 1) reconstruction of the state $D(E(s_i)) = s_i$, 2) reconstruction of the reward $R(E(s_i)) = r_i$ and reconstruction of the successor state $T(E(s_i), a_i, x_i) = E(s_{i+1})$.

Once the system is trained, Dreamer is capable of generating action and latent code sequences in its "head" without interacting with the real environment, hence the motivation for its name.

Dreamer for leanCoP: State Representation [4] only use training signal coming from reward reconstruction (predicting if a rewrite is valid), while [1] show that most of the representational power of Dreamer comes from state reconstruction. However, in the case of theorem proving, reconstructing the state is not as straightforward as for ATARI games, as the former has discrete proof objects, which requires some hand-crafted mapping before processing by neural models. We consider three methods for extracting features from leanCoP states: 1) manual features developed in [3] and successfully used in many systems, e.x. [2, 11], referred to as *ENIGMA features* and 2) using the graph neural network developed in [5] specifically for logic formulae, referred to as *GNN* and 3) *transformer* language models that process text directly.

ENIGMA features rely on consistent naming of concepts, while the GNN exploits structural similarities of terms, so they are somewhat complimentary. Transformers are recently gaining attention in theorem proving (e.g. [9, 7]). Hence, both the input of the encoder E and the output of decoder D can be either ENIGMA features, a graph or plain text. We intend to experiment with all combinations to see what results in the best latent space structure.

Dreamer for leanCoP: Action Selection Dreamer assumes either a fixed size discrete action space or a real n-dimensional space, neither of which matches the action space of a theorem prover, since the valid actions are state dependent. However, in leanCoP, the set of literals contained in the input clauses constitute a fixed size superset of valid actions, so we can let our model select from this superset. All actions are encoded using either ENIGMA features, graph embedding or plain text and we train an action encoder $A_e : A \to \mathbb{R}^n$. The transition model concatenates latent state and action codes before predicting the successor state.

Our current implementation uses the same graph network for states and actions, which contains nodes for each clause derived either from the tableau or from the actions. In the case of state embedding, we collapse every clause while in the case of the action embedding we extract only the relevant part of the graph for each action.

Note that different axioms yield different possible actions, hence the action space can vary across problems. For this reason, our current architecture does not support model building for heterogeneous problem corpora; it is instead suitable for modeling problems within a single theory. We argue that this is the setup in which a world model makes most sense.

Dreamer for leanCoP: Sample Selection Dreamer maintains a buffer of previously explored episodes (proof attempts) which it samples from in each training step. In order to make the length of samples uniform, each sample is a fixed length slice of an episode. However, the length of proof attempts can vary greatly across problems, so we introduce a length balancing mechanism. We maintain a set of buckets $b_1 \dots b_k$ where b_j holds slices of length l in the range $2^{j-1} < l \leq 2^j$. During replay, we generate 2^{M-j} samples from bucket b_j where 2^{M-j} is the batch size. This ensures that samples from episodes of different length contribute equally. To make the sampled data more balanced, 20% of the samples are chosen with a positive reward sum to compensate for the fact that most proof attempts fail and yield no positive reward.

Dreamer for leanCoP: Rewards and Losses Assigning rewards to theorem prover actions is a well known challenge. A specialty of latent space reasoning is that we do not know what inferences are valid in a particular state, so the model has to learn that as well. To aid this, we

give high negative reward for invalid moves. Our reward function R for action a performed in state s is:

$$R(s,a) = \begin{cases} 1 & \text{if a proof was found} \\ -0.2 & \text{if there are no more valid moves} \\ -1 & \text{if } a \text{ is an invalid move} \\ -0.5 & \text{if a step limit is reached} \end{cases}$$

During training, we have the convenience that we know which actions are valid. We exploit this by adding an extra loss term to the world model, which is the negative log probability of choosing a valid step. This helps to make convergence faster.

Current Status We created an RL environment that encapsulates leanCoP and adapted the Dreamer codebase to the particularities of the environment. We implemented ENIGMA feature extraction as well as the graph model, but we have not started working on the transformer model yet.

We are running first experiments, tuning hyperparameters and evaluating the consistency of the latent representation. We find that the model struggles with the sparsity of the rewards, even when training slices are balanced. While the model quickly finds ways to avoid illegal moves and failure states by infinite derivations, it cannot yet make good use of the positive signal coming from successful proof attempts.

Conclusion Our project explores the possibility of building a world model for an automated theorem prover that captures its internal dynamics. We adapted the DreamerV2 architecture to the leanCoP connection tableau calculus and started running first experiments. The promise of such world models is to yield a semantically meaningful latent structure, in which one can identify promising directions, leading to better exploration and more targeted proof search.

Acknowledgments This work was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002), the Hungarian National Excellence Grant 2018-1.2.1-NKP-00008 and by the Hungarian Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program.

References

[1] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[2] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olsák. Reinforcement learning of theorem proving. In *NeurIPS*, pages 8836–8847, 2018.

[3] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient semantic features for automated reasoning over large theories. In Qiang Yang and Michael Wooldridge, editors, *Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 3084–3090. AAAI Press, 2015.

[4] Dennis Lee, Christian Szegedy, Markus Rabe, Sarah Loos, and Kshitij Bansal. Mathematical reasoning in latent space. In *International Conference on Learning Representations*, 2020.

[5] Miroslav Olsák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on*

Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020), volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 1395–1402. IOS Press, 2020.

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

[7] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020.

[8] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019.

[9] Josef Urban and Jan Jakubuv. First neural conjecturing datasets and experiments. In Christoph Benzmüller and Bruce R. Miller, editors, *Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings*, volume 12236 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2020.

[10] Zsolt Zombori, Adrián Csiszárik, Henryk Michalewski, Cezary Kaliszyk, and Josef Urban. Towards finding longer proofs. *CoRR*, abs/1905.13100, 2019.

[11] Zsolt Zombori, Josef Urban, and Chad E. Brown. Prolog technology reinforcement learning prover. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 489–507, Cham, 2020. Springer International Publishing.

[12] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020.