Scaling Interpretable Reinforcement Learning via Decision Trees

Ellis Brown
elbrown

December 10, 2021

Abstract

Deep reinforcement learning is a powerful tool for learning complex control tasks; however, neural networks are notoriously "black boxes" and lack many properties desirable of autonymous systems deployed in safety critical environments. In this project, we focus on methods that result in a final control policy specified via a decision tree—which is thus interpretable and verifiable. We build upon a prior method, VIPER, that first learns a high-performing "expert" policy via any standard Deep RL technique, and then distills the expert policy into a decision tree. Our method, called MSVIPER, is specifically designed to scale to complex environements that greatly benefit from (or require) curriculum learning to be solved; we leverage the structure in the currculum stages to enable more efficient learning and a smaller (and thus more interpretable) decision tree. To demonstrate the ability of our method to succeed in complex environments, we apply it to Minecraft—a challenging open-world environment. We highlight that our method is amennable to post-training verification and modification or improvement.

1 Introduction

The last decade or so has seen a boom in Reinforcement Learning (RL), largely spurred by the application of growing advances in deep learning. Deep RL has been successfully applied to solve various simulated tasks [12, 15] with superhuman ability, and has great potential to transform various real-world control tasks in the coming years. Many of society's most safety-critical tasks are performed by humans, such as driving, air traffic control, and emergency response. If deep RL could achieve superhuman performance on such important real-world tasks, it would constitute a substantial advance for modern society.

The success of deep learning is due to the application the massive amounts of computational power and data available today to Neural Networks (NNs). NNs are made up of layers of several "neurons" (perceptrons) whose activations are passed through a nonlinear function and fed to each neuron in the subsequent layer. An input-output mapping is learned via the weightings between neurons using data. This structure can be extended both in terms of the number of layers and the number of neurons in each layer to capture complex relationships; notably, NNs are known to be universal function approximators. The complex structure that gives rise to this great representational power makes it nearly impossible to understand the inner workings that lead to an output; they are notorious "black boxes." This does not bode well for their application in our desired safety-critical situations, which require properties such as, interpretability, verifiability, and robustness guarantees to be properly trusted. A growing concern is that autonomous systems based upon NNs that are deployed in the real-world will introduce (potentially immense) risk due to these shortfalls.

Of course there is enormous interest in address these limitations. There is a growing body of work aiming to do so generally and in RL. However these efforts are still in their infancy, and the world is increasingly pushing towards the deployment of autonomous systems—just look at the buzz around self-driving today. We are interested alternative approaches to attempting direct interpretation of neural network policies themselves; specifically, we focus on approaches that result in policies represented by decision trees.

Decision trees are one of the most widely understood "interpretable" methods in machine learning, and are widely regarded for the various properties we desire of safety-critical systems. Decision trees have similar representation power to NNs; they are nonparametric, and can be extended to arbitrary depth to represent arbitrary functions (though we prefer smaller trees for our purposes, as they are more interpretable). The main challenge with using decision trees is that they are significantly more dificult to train than NNs; making it very difficult to practically arrive at a successful control policy.

One recent approach called VIPER [2] aims to leverages the learning abilities of NNs and result in a similarly capable final policy represented by a decision tree—thus overcoming the primary challenge of using decision trees in RL (see (2.1.1)). VIPER's benefits have been displayed on toy problems such as Cartpole [4] and Atari Pong [2], but it is decidedly still a proof-of-concept; it has unfortunately not yet been successfully applied to moderately complex environements (much less those with comparable complexity to the real world).

Taking this as inspiration, MSVIPER is an extension of VIPER explicitly designed to scale to complex environments that require curriculum learning to be solved (see (3.1)). The curriculum is first used to learn the expert policy using any RL method; the currculum is then again used to produce a better performing and smaller tree policy, with lower sample complexity than standard VIPER.

In order to display the ability of MSVIPER to scale to complex environements, this project seeks to apply it to Minecraft—a challenging environment with sparse rewards and many innate task hierarchies and subgoals (see 2.2). Players in Minecraft navigate a 3D open-world environment in first-person, pursuing various goals primarily centered around collecting resources and crafting increasingly complex items. Tasks defined in Minecraft present a challenge several orders of magnitude more complex than the toy problems VIPER has been applied to. Parallel work applies this methodology to robot navigation in real-world crowds.

2 Background

2.1 Interpretability

There are a number of recent approaches towards interpreting [6] and verifying [10] NNs. This work is still largely limited, however NNs are often still often preferred due to their performance and broad applicability. Recent work shows that their performance advantages over other methods is not due to better representative power, but rather simply the fact that they are better regularized and therefore easier to train [1].

Decision trees are one of the primary tools used in situations where interpretability is paramount. Their tree structure is inherently interpretable to humans, and they are able to capture much more complex relationships between variables than other popular "interpretable" methods can (e.g., linear models). Previous work has used decision trees in conjunction with RL. The challenge with decision trees is they are difficult to train, espcially in complex settings like RL.

2.1.1 VIPER

We choose to build upon VIPER[1] [2], which combines ideas from model compression and imitation learning to provide great flexibility and final interpretability advantages. Unlike other methods, VIPER allows the use of *any* method to learn the expert policy—allowing us to take advantage of the benefits of NNs. Furthermore, it results in a *single* decision tree, which has benefits in terms of interpretability and verifiability.

In VIPER, a neural network expert policy is first learned to solve the desired task. This expert is then used to learn a "good" decision tree policy through an extension of the popular DAGGER imitation learning algorithm [13] that utilizes the expert's Q-value estimates. VIPER learns relatively small decision tree policies (< 1000 nodes) that exhibit strong performance on a number of toy examples.

Unfortunately, VIPER has only yet been applied to such toy examples and fails to scale to more complex tasks. Though it has several properties attractive for our use case, it is far from being ready for usage in the real world.

2.2 Minecraft

Minecraft serves as a rich and challenging setting for RL. It is a complex open-world environment with intricate task hierarchies and subgoals centered around obtaining resources and crafting items.

2.2.1 MineRL

A group of researchers at Carnegie Mellon have created a flexible framework to define RL environements and tasks for Minecraft called MineRL [7]. MineRL interfaces with the Malmo Minecraft framework [9] to construct and control a Minecraft simulation, and integrates with the widely used OpenAI Gym RL framework [5]. It provides functionality to specify custom environements, with control over aspects such as terrain characteristics, spawn location and inventory, and reward schedule.

3 Methods

3.1 MSVIPER

MSVIPER[2] is an extension of VIPER that is specifically designed to scale to complex environments that greatly benefit from (or require) Curriculum Learning (CL) to be solved. In the first phase, CL is leveraged to enable an expert policy to be learned via any RL method. Our expert policies are trained using PPO [14] or A2C [11] The second phase leverages the inherent structure in the curriculum stages to improve upon VIPER's decision tree policy distillation. MSVIPER is more sample efficient and outputs a "better" tree policy—that is better performing and smaller.

3.1.1 Learning via Curriculum

To enable better learning and generalization improvements, CL introduces examples to learning algorithms in a meaningful order that gradually illustrates more concepts, and gradually more complex ones [3].

[1]Verifiability via Iterative Policy ExtRaction

[2]Multiple Scenario VIPER

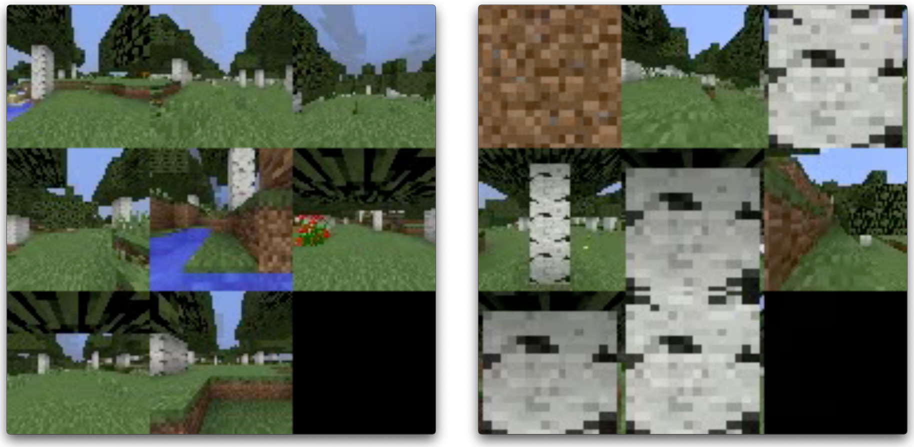

(a) Agents initialized in the environment (b) Agents reaching trees

Figure 1: POV of eight Minecraft agents during training of the Reach Tree curriculum stage. In (a), we see the viewpoint of the agents just after initialization in the environment. In (b), we see that the agents have learned to reach trees—although not cut them down, yet.

To train the expert agent using CL, we trained in a succession of sub-tasks (defined as Gym environments) of increasing difficulty rather than just once on the final task itself. The policy learned in each sub-task is reused as the initialization for the subsequent stage. By breaking the task into well-scoped sub-tasks, CL makes it easier to learn a good policy. As noted earlier, the primary challenge with decision tree policies is that they are difficult to train, so this property is also particularly useful in the next phase.

3.1.2 Distillation to a Decision Tree

MSVIPER improves the process of distilling the expert policy to a decision tree policy using inspiration from CL. At a high level, this phase consists of generating trajectories (sequences of state-action pairs) and then training a decision tree policy on the trajectories using imitation learning.

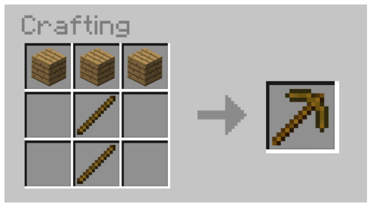
Applying ideas from CL to imitation learning is not straightforward. Imitation learning uses expert demonstrations from a single task (here, the final) to train a policy. However in our case, we have the expert policy and each of the curriculum stages it was trained upon! We use the *final* expert to generate imitation learning trajectories in each of the curriculum stages used to train it in the first place. We then train the decision tree policy on a *curriculum of trajectories* from each stage. The variety in this curriculum and the simulation environment itself ensures that the decision trees that are trained on trajectory samples are more robust to various scenarios that might be encountered at test time.

3.2 Task: Obtain Wooden Pickaxe

The task we designed is to obtain a wooden pickaxe after being spawned in a forest with no items. Agents can craft simple items in a 2×2 grid on their own; they must craft a crafting table and use it to craft more complex items in a 3×3 grid. The crafting recipe is depicted in Figure 2.

1. Craft 12 planks using 3 logs
2. Craft 1 crafting table using 4 planks
3. Craft 4 sticks using 2 planks (via crafting table)
4. Craft 1 wooden pickaxe using 2 sticks + 3 planks (via crafting table)

(a) Wooden Pickaxe Crafting Recipe



(b) 3 × 3 crafting grid

Figure 2: Crafting a Wooden Pickaxe

3.2.1 Curriculum

The only naturally occuring resource an agent needs to acquire to achieve this task is logs, which are obtained by chopping down trees (facing trees and attacking for several seconds). The curriculum stages for the Obtain Wooden Pickaxe task are listed in Figure 3.

1. Navigate to trees (see Figure 1)
2. Chop down trees to obtain wood
3. Craft a crafting table
4. Use table to craft sticks
5. Use table to craft a wooden pickaxe

Figure 3: Curriculum Stages

3.3 Results

Work on this project has gone towards (i) using MineRL to design Gym environments for each sub-task (each with their own defintion, rewards schedule, etc.), (ii) constructing preprocessers to transform the raw observations from the MineRL engine into a form easily consumable by a decision tree, (iii) and on training expert neural network policies to solve the tasks. Mean reward from a training run on the first curriculum stage are displayed in Figure 4.

As it turns out, hand designing *good* environments (with well-shaped rewards) is much easier said than done. The neural network policies turned out to be good at finding and exploiting imperfections in environment specifications that gave them reward without necessarily solving the task ("reward hacking" [8]). Unfortunately training policies here is a computationally heavy endeavor. We train agents for 2 million gradient steps, with 8 agents collecting training samples in parallel, which takes a around 8 hours per run (making the iteration process relatively delayed). After several iterartions of these three phases, we were unfortunately unable to get to the second phase of MSVIPER—distillation to tree policy.

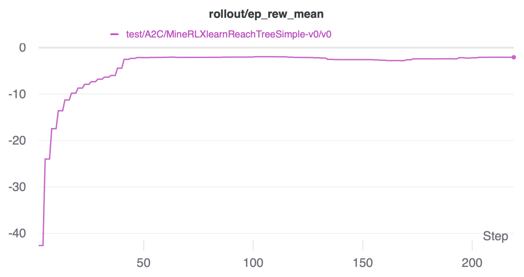


Figure 4: Training curve from an A2C expert agent on the tree-reaching curriculum stage. Each step on the graph is 10,000 gradient steps.

Acknowledgements

Aaron M. Roth, a former CMU MSR student and current UMD PhD student, conceived of MSVIPER and showed initial results in robot navigation. EB would like to thank him for his mentorship and advice throughout this project.

References

[1] J. Ba and R. Caruana. Do deep nets really need to be deep? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[2] O. Bastani, Y. Pu, and A. Solar-Lezama. Verifiable reinforcement learning via policy extraction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[3] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[4] S. Bhupatiraju, K. K. Agrawal, and R. Singh. Towards mixed optimization for reinforcement learning with program synthesis. July 2018.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. June 2016.

[6] F. Doshi-Velez and B. Kim. Towards a rigorous science of interpretable machine learning. Feb. 2017.

[7] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, et al. The MineRL competition on sample efficient reinforcement learning using human priors. *NeurIPS Competition Track*, 2019.

[8] D. Hadfield-Menell, S. Milli, P. Abbeel, S. Russell, and A. Dragan. Inverse reward design, 2020.

[9] M. Johnson, K. Hofmann, T. Hutton, D. Bignell, and K. Hofmann. The malmo platform for artificial intelligence experimentation. In *25th International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI - Association for the Advancement of Artificial Intelligence, July 2016.

[10] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer. Algorithms for verifying deep neural networks, 2020.

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

[13] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Proceedings of the fourteenth*, 2011.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. July 2017.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan. 2016.