A Framework for Formal Verification to Correct Actions in Reinforcement Learning

Ethan Hobbs Vikas Nataraja

April 11, 2020

Abstract

In reinforcement learning, proving the possibility of a safe state is inherently difficult due to the range of methods available to find a new safe state. One of the most common options for finding a new state when the agent reaches an unsafe one is reverting to an initial state. In many cases, these methods, like the one mentioned above, are inefficient or might not actually produce a safe state transition. In this report, we propose a new method for verifying an agent's policy for transitioning out of an unsafe state that will always find a new safe state to transition to.

1 Introduction

Reinforcement Learning (RL) has gained immense momentum in recent years particularly in the field of robotics where tasks are repetitive and RL can make an instant impact. This is because the agent can learn a policy that maximizes the reward function much quicker in repetitive tasks because the reward environment is denser and guarantees near-continuous rewards for every action that the agent takes. With RL gaining popularity, verification of such systems is an active area of research in Computer Science. The core problem of any software verification is to verify that a given system satisfies its specification. A conventional way to verify software in general is to establish safety rules before the agent is deployed in the environment which requires extensive data [1]. It is not always possible to predict the states or the changes in the environment beforehand particularly if it is dynamically changing. Another common approach that is more widely deployed in Machine Learning is for the system itself to verify its own progress [2, 3]. While it is difficult to characterize such a verification, it affords better safety which is essential in RL (and relational verification of RL) which is more transparent than other machine learning fields [4] . Formal state verification then becomes essential so that the agent can monitor its progress by checking the validity of states.

2 Overview

Consider a reinforcement learning system like an inverted pendulum or a cartpole. In our implementation of the system, we synthesize a deterministic program that approximates the neural network policy developed by the reinforcement learning algorithm. We can then verify the deterministic version of the program and use it to shield the actual neural policy from entering into an unsafe state. In the verification, we look at a transition from state a to state b dependent on whether state b is "safe." Safe in the context means that the state is reachable by the reinforcement learning

system and that it will not violate the failure conditions set out in the reinforcement system. While the verification process gives us a good idea on what is a safe state, it will not guarantee it since the synthesized program is an approximation. We then have to perform a second shielding step at runtime to prevent any edge cases. This method described above has previously been explored in [2].
In the safety verification algorithm of [2], they reduce the diameter of the state space by a factor of 0.5 if a counterexample is found. Since a reinforcement learning systems like a robotic arm are physical, it would be desirable to create a more representational state space transformation rather than the simple reduction. If this is done, an operations like restoring to the previous safe state could be performed if a counterexample is found in the transitions of the current state.

3 Related Work

In recent years, significant work has gone into verification of reinforcement learning and more specifically, the verification of states. A two-pronged system was implemented in [2] where the concept of safety and safety verification were both baked into the synthesis of a deterministic program with an inductive invariant alongside the neural policy. However, when an invalid or incorrect state is observed, the agent reverts to one of initial safe states. In our approach, the agent back propagates to an already traversed safe state. A similar formal verification approach was done by [3] but to verify the actions of an autonomous robot by constructing a finite state abstraction and performing reachability analysis over the search space. However, if the initial set of safe states are not within a certain limit, there is a high risk of state space explosion. At the same time, it does not apply to unbounded states and does not capture the relationships between the state variables. These shortcomings are bypassed in our approach because we use the existing state space to find a safe state. [5] introduced a method to certify deep neural networks by using a novel abstract interpreter which relies on a combination of floating-point polyhedra and activation functions such as ReLU (Rectified Linear Unit) and the sigmoid function. While this approach works to prevent exploding gradients during training, it works by transforming activation functions which inherently means that the neural network, in its current form, cannot be made robust to deeper architectures. In our proposed framework, we eliminate the need for changing activation functions by solving for state space. [6] focused on relational verification represented as a Markov Decision Process (MDP) solvable via reinforcement learning. The most common problem with MDPs is that high-dimensional state spaces are not solvable. In our proposed method, we aim to use the existing policy to find a safe state which does not expand the dimensionality. A novel reinforcement learning framework to represent policies as high-level programming language capable of generating interpretable and verifiable agent policies was presented by [4]. This framework works when the agent's policy is deterministic which naturally inhibits non-deterministic policies. In our method, we propose to use a stochastic policy which expands our application space.

References

[1] Divya Gopinath, Guy Katz, Corina S. Pasareanu, and Clark W. Barrett. Deepsafe: A data-driven approach for checking adversarial robustness in neural networks. *CoRR*, abs/1710.00486, 2017.

[2] He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 686–701, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, page 147–156, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning, 2018.

[5] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), January 2019.

[6] Jia Chen, Jiayi Wei, Yu Feng, Osbert Bastani, and Isil Dillig. Relational verification using reinforcement learning. *Proc. ACM Program. Lang.*, 3(OOPSLA), October 2019.