A Domain-Specific Language for Multi-Agent Reinforcement Learning in Distributed Systems

Tim Molderez, Bjarno Oeyen, and Wolfgang De Meuter

Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium
{tim.molderez, bjarno.oeyen, wolfgang.de.meuter}@vub.be

Abstract. Many multi-agent reinforcement learning (MARL) problems are well-suited to be solved in a distributed manner. Not only to achieve scalability in terms of runtime or memory, but also because the environment may be a distributed system. While distributed machine learning is appealing in these cases, it does introduce several complexities inherent to distributed computing This paper presents Marlon, a domain-specific language to facilitate MARL in a distributed environment. Marlon essentially acts as a bridge between the environment and the agents observing it, and abstracts away many of the concerns related to distributed computing. We demonstrate Marlon by means of a load balancing example, where MARL is used to optimize the system's performance, while allowing network nodes to join and leave the system.

Keywords: Domain-specific languages, multi-agent reinforcement learning, distributed systems

1 Introduction and overview

This work targets the use of multi-agent reinforcement learning (MARL) in a distributed context. That is, our primary focus is on the use of MARL where the environment is a distributed system, in a broad sense of the term. For instance, it includes applications intended for smart grids, traffic control systems, wireless sensor networks, or vehicle-to-vehicle communication networks. Such systems may involve a large number of networked devices. Due to their scale, it is not trivial to ensure such systems perform optimally in terms of e.g. throughput, reliability, latency or resource usage. Observing that each node in the network is an autonomous entity, MARL would be a natural fit for tackling this type of optimization problems. However, the use of MARL is complicated by a number of factors inherent to distributed systems: nodes may dynamically join/leave the network, lose their connection or even crash entirely.

To allow MARL researchers to focus on their area of expertise, and abstract away the different considerations that need to be made for distributed systems, we developed a domain-specific language called Marlon[1][2]. The same also holds

[1] Marlon can be found at: https://gitlab.soft.vub.ac.be/smileit/marlon-dsl
[2] A video demonstration is available at: https://youtu.be/dzM9CJT2oSU

from the perspective of distributed system experts; Marlon facilitates the integration of existing MARL algorithms in distributed systems without the need to know the intricacies of machine learning. In short, Marlon acts as a bridge between the two domains of MARL and distributed systems. The language is built on top of the Elixir host language, which is designed for developing large distributed systems.

A Marlon program consists of two key components: *actors* and *goals*. The distributed system, i.e. the environment, is composed of actors. This notion of actors stems from the actor concurrency model [1]. Each actor runs in its own thread of execution, has its own isolated state and can only communicate with other actors by sending messages. The interface between this actor-based environment and a MARL algorithm is specified with a goal. A goal in Marlon specifies all the information required to configure a MARL algorithm: the reward function, the action space, which MARL algorithm to use, the algorithm's parameters, which information should be shared among agents, etc. To effectively connect a distributed environment to a goal specification, three kinds of statements should be inserted in the environment: first, a statement that attaches an agent to an actor, such that the agent can observe the actor's state and let this agent perform actions. Second, a statement to indicate *when* the agent must choose an action. Third, a statement to indicate when the reward should be computed. Finally, Marlon provides an API that makes it possible to reuse existing MARL algorithms, or add new ones. Given that the Elixir host language is not commonly used for machine learning, it also is possible to interface with MARL algorithms written in Python or C++.

2 Demonstration

To showcase the flexibility of Marlon, and to demonstrate the language both from the perspective of a MARL researcher and and distributed systems expert, we have implemented the load balancing use case that was also used to evaluate the ESRL MARL algorithm presented in the work of Verbeeck et al. [2]. This use case involves a master network node that distributes a given job across a number of heterogeneous worker nodes. Workers are free to join or leave the network at any time, and they can request a certain amount of work from the master. MARL is used to optimize this amount for each worker.

References

1. Agha, G.A.: Actors: a model of concurrent computation in distributed systems (1986), http://cds.cern.ch/record/1220706
2. Verbeeck, K., Nowé, A., Parent, J., Tuyls, K.: Exploring selfish reinforcement learning in repeated games with stochastic rewards. Autonomous Agents and Multi-Agent Systems **14**(3), 239–269 (Jun 2007). https://doi.org/10.1007/s10458-006-9007-0, https://doi.org/10.1007/s10458-006-9007-0