Rlgraph: Modular Computation Graphs For Deep Reinforcement Learning

Paper by Michael Schaarschmidt, Sven Mika, Kai Fricke, Eiko Yoneki

Presentation by Matthew Guest

Reinforcement Learning

- Has its origins with Markovian Decision Processes (Bellman, 1957)
- Q – value vs Q - learning
- Deep Q-Network Deepmind (Volodymyr Mnih et al, 2013)
- DNQ Issues: Forgetfulness / Volitility; Enormous state-space
- Algorithmic Progress: Dueling DNS (Ziyu Wang, 2015), IMPALA
- Environment Standardisation: OpenAI gym
- Reinforcement Learning Frameworks…

Challenges of Reinforcement Learning

- No Supervision, Feedback is Delayed (Credit Assignment Problem)
- Data observed is causally effected by agents' actions.
  - Therefore, Actor / Environment feedback loop is sequential
  - Parallelisation (Especially if environment not simulated)
- Non-determinism of environment and stochastic nature of many approximations cause issues with testing and reproducibility.
- Large Search Space -> Computational Power:
  - Seeking to benefit from distributed approach

RL Frameworks

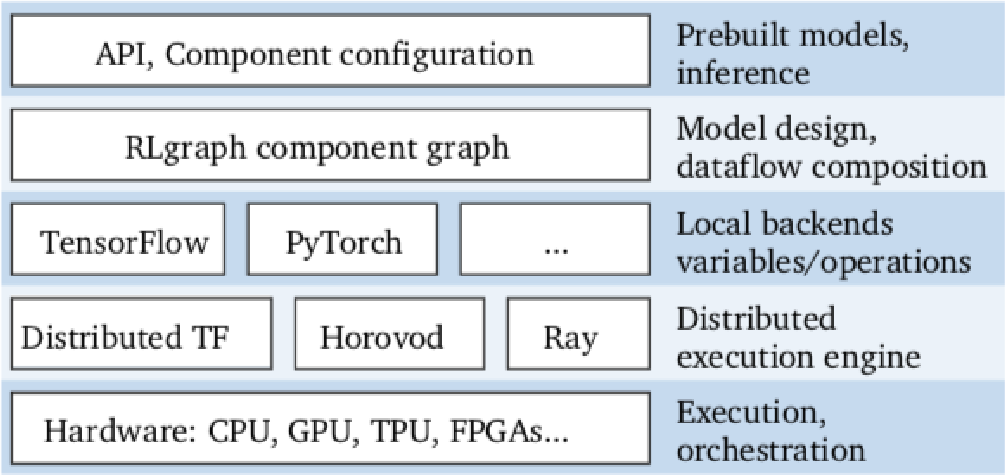
- OpenAI gym – Baselines for testing, Environments (Atari, 3D, …)
  - Tool for environments, libraries favour conciseness over extensibility
- TensorForce – TensorFlow library for Deep RL
  - Declarative API
  - Modular Components (not decomposable)
  - Algorithm and Application Separation
- Ray RLib – RL Library for Ray Distributed Execution Engine
  - Distributed RL Library. Native to Ray with Central Command Framework

Rlgraph Overview

- This paper outlines a new, unifying Framework with the aim to improve:
- Incremental Building Testing
  - To improve the speed of prototyping and robustness of production models.
- Distributed Execution
  - By focusing on modularisation, RLgraph aims to separate the concerns of design and execution.
- Extensibility
  - By separating “logical component composition, backend graph definition and distributed execution,” components are interchangeable and well defined.

Rlgraph Components

- The Rlgraph framework is primarily a Component graph.
- A Component class can encapsulate arbitrary computations.
- A Component contains internal methods, API methods, variables, and associated sub-components.
- This graph structure is an abstraction that can be executed across platforms.



Framework Design

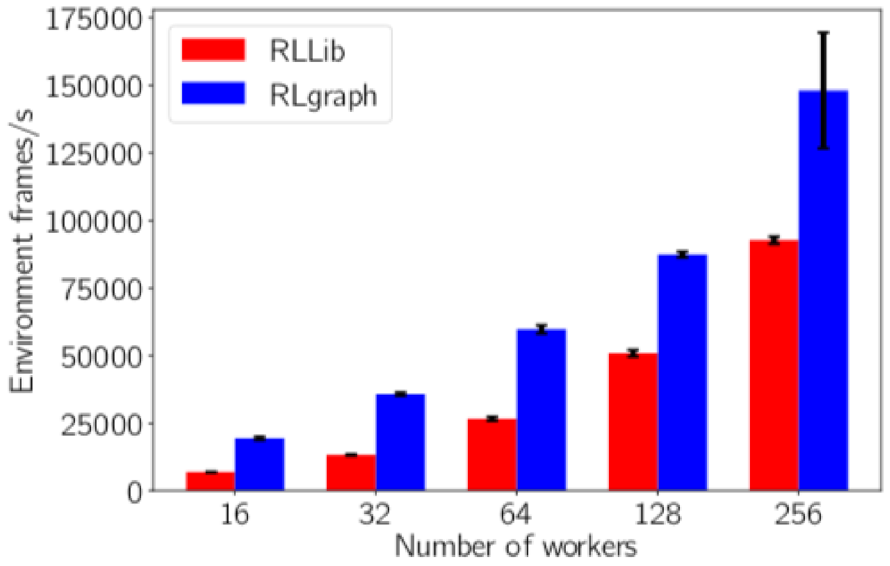
- Separating algorithms and execution
  - RL algorithms require complex control flow to coordinate distributed system and sample collection.
  - Agents’ policies require internal training logic.
  - Components separate concerns.
- Reusable components with strict interfaces.
  - Interchangeable components, not dissimilar to NN layers in Keras for example.
  - Components interact only via strict, declared APIs. Static analysis
- Incremental sub-graph testing.
  - Components in Rlgraph may be individually built and tested.

Identified Failings of Existing Frameworks

- TensorForce – TensorFlow library for Deep RL
  - Modular, but lack Modular, Separated Build Process for Testing
  - Unnecessary context switching between TF runtime and Python interpreter
- Ray RLib – RL Library for Ray Distributed Execution Engine
  - Lacks portability due to Ray nativity
  - Restricted Control Flow inherited from Ray

Results

- Build overhead: Sub 1 second for both TF and PT
- Worker Performance Baseline: No overhead on TF, Slight overhead PT
- Distributed execution on Ray:
  - Outperformed Rlib in Ray environment



Follow on

- This paper was 2019
- RL Algorithms: Multi Agent & Semi Supervised
- Frameworks: Acme (Deepmind June 2020)

Q&A