Modular-RL: A Flexible Framework for Deep Reinforcement Learning

Jonathan Lamontagne-Kratz

McGill University, School of Computer Science

Motivation

Reinforcement Learning (RL) research often suffers from rigid codebases that are difficult to adapt to new environments or novel algorithm combinations. **Modular-RL** was developed to solve this bottleneck.

Core Objectives:

- **Agile Development:** Enable rapid iteration and testing of RL models.
- **Broad Applicability:** Easily apply State-of-the-Art (SOTA) models to new game types and environments.
- **Algorithmic Flexibility:** Seamlessly test combinations of advanced agent architectures and search methods.

Framework Architecture & Decomposition

A critical innovation of this framework is the complete **modularization of agent definitions**. Instead of monolithic agents, we decompose an agent into its constituent parts, defining it abstractly through output heads, action selection methods, and losses. This approach also applies to the training loops, separating the logic of state management and update steps.

Agent Decomposition

A specific agent (e.g., MuZero) is defined by configuring four key modules. This approach makes the agent shell otherwise abstract:

- **1. Output Heads:** Defining the final network layers (e.g., Actor Head, Value Head).
- **2. Action Selection:** The logic applied after prediction (e.g., MCTS, greedy argmax).
- **3. Loss Functions:** Specifying which losses define the agent's gradient updates.
- **4. Agnostic Training Loop:** A loop that coordinates learning but doesn't contain agent-specific logic.

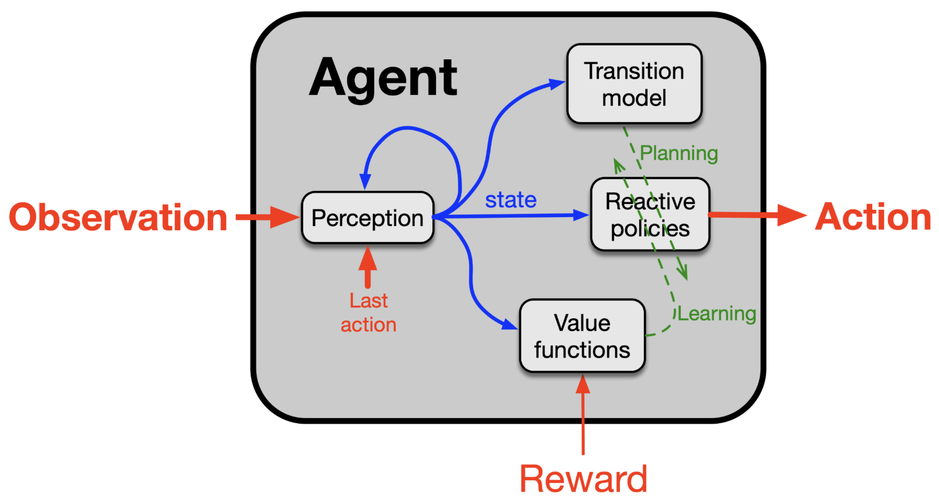

Fig. 1: Overview of Modular Agent Architecture

Key Software Features

- **Decoupled Components:** Neural architectures (e.g., ResNets, Transformers) are strictly separated from the RL algorithms.
- **PufferLib Integration:** Built to support highly optimized parallel environment execution.
- **Agnostic Training Loops:** The loop orchestrates interactions with the replay buffer, data batching, and applies the agent-specific loss functions, making it reusable across different agents.

Going Beyond Google's `mctx`

A major architectural achievement of Modular-RL is its handling of complex game logic, specifically improving upon standard libraries like DeepMind's/Google's `mctx`.

While `mctx` is highly optimized for standard, deterministic, alternating-turn games (like Go or Chess), Modular-RL introduces natively supported abstractions for:

- **Complex Player Turn Dynamics:** Safely handling multi-agent environments with variable turn orders or simultaneous actions.
- **Stochastic Games:** Extending tree-search capabilities to environments containing chance nodes or hidden information.
- **Modular Search Algorithms:** The ability to cleanly swap and combine Search algorithms (e.g., standard MCTS vs. Batched MCTS) without rewriting the core RL agent.

Agent Validation & Results

To validate the modular architecture, we implemented SOTA algorithms (such as **MuZero**) entirely using the framework's modular components and benchmarked them against classic environments.

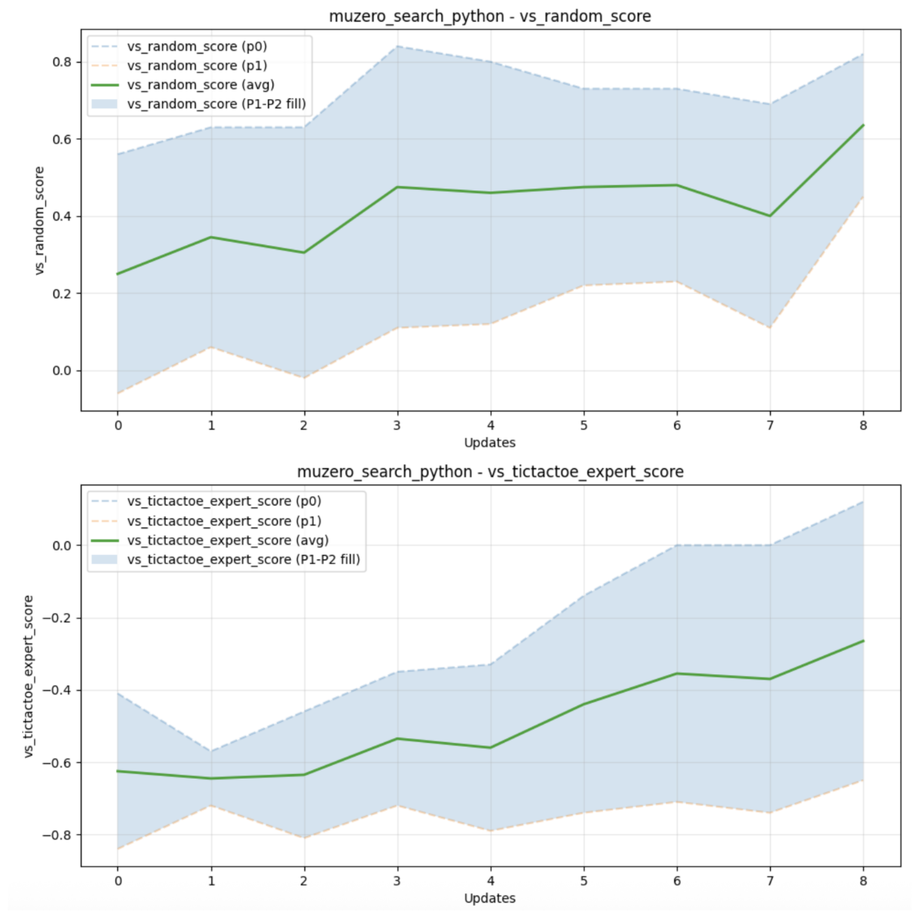

Fig. 2: MuZero Performance (UCB) on Tic-Tac-Toe

Findings: The framework successfully trains complex agents. As seen above, the MuZero implementation quickly achieves optimal Upper Confidence Bound (UCB) performance against the Tic-Tac-Toe expert, validating that the decoupled MCTS and neural network modules interact correctly.

Future Work

With the core architecture stabilized and successfully tested on smaller environments, the next phase of development focuses on scale and complexity.

1. Tackling High-Complexity Environments

Applying the current agents to **Catan** to leverage the framework's stochastic game and complex turn-dynamic capabilities.

2. Best-Practice Scaling

- **Cluster Deployment:** Optimizing the code for distributed training on multi-node computing clusters.
- **Advanced Memory Backends:** Incorporating advanced, highly concurrent replay buffer libraries for faster data ingestion.
- **Centralized Inference:** Separating actor workers from learner nodes to maximize GPU throughput and reduce rollout bottlenecks.

Conclusion

Modular-RL demonstrates that it is possible to build an agile, extensible Reinforcement Learning codebase without sacrificing the ability to implement deep, complex SOTA algorithms. By overcoming the limitations of rigid monolithic implementations, researchers can rapidly prototype next-generation agents across a wider variety of stochastic and multi-agent games.