Rust Reinforcement Learning Framework

Overview

A modular and extensible reinforcement learning framework written in Rust, supporting algorithms such as Q-Learning, SARSA, MCTS, AlphaZero, and MuZero. This project allows benchmarking across different environments like `CartPole` and `MountainCar` with a unified interface.

Description of the Algorithms and Theory

This project implements the following reinforcement learning (RL) algorithms.

(For more information, see the RL book: `http://incompleteideas.net/book/the-book.html` or AlphaZero paper: `https://arxiv.org/abs/1911.08265`)

- **Q-Learning**: A model-free off-policy RL algorithm that learns the value of action-state pairs (Q-values) using the Bellman equation. It chooses actions greedily based on learned Q-values.
- **SARSA**: An on-policy variant of Q-Learning. It updates its Q-values using the action actually taken, rather than the maximum possible next action. (In contrast to Q-Learning, which always assumes the agent will act optimally.)
- **Monte Carlo Tree Search (MCTS)**: A search-based planning algorithm using tree expansion guided by simulations. It builds a tree using selection, expansion, simulation, and backpropagation to evaluate actions in large state spaces. Not a learning algorithm per se.
- **AlphaZero**: A deep RL algorithm that integrates MCTS with a neural network-based value and policy model, trained from self-play, improving over time with gradient-based learning. The original AlphaZero uses a shared network with a value and a policy head. The policy head is used inside the MCTS to give priority to better nodes without needing to explore them. In my experiments this didn't lead to good results, so I just use the value head.
- **MuZero**: An extension of AlphaZero that does not require a model of the environment's dynamics. Instead, it learns a latent space representation and internal transition model from raw observations, combining planning with model learning. Due to training difficulties steming from the compexity of this algorithm, this implementation is a full MuZero, but only partial. The dynamics network is not used in the MCTS, since I could not get it to train properly.

Description of the Implementation

The project is structured into following modules:

- `environment`: Trait-based interface for RL environments (`CartPole`, `MountainCar`) that handle step logic and state transitions.
- `agent`: Abstractions and concrete implementations for all agent types.

- `logger`: Handles training logs, reward tracking, data export, and plotting.
- `rl_agent`: Shared agent behavior like training, benchmarking, and evaluation.
- `qlearning`, `sarsa`, `mcts`, `alphazero`, `muzero`: Algorithm-specific logic and data structures.
- `main.rs`: Entry point handling CLI arguments, initializing environments/agents, and managing training and evaluation.

Each agent conforms to a common `Agent` trait, ensuring interchangeable benchmarking and training pipelines.

Environments

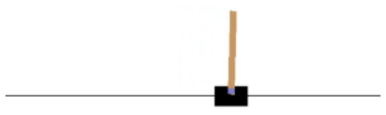
The environments are based on the standard RL gymnasium used for benchmarking (https://gymnasium.farama.org/). The specific implementation is a fork of the RL gym for Rust and modified for my use case.

The framework currently supports the following environments:

- `CartPole`

  `https://gymnasium.farama.org/environments/classic_control/cart_pole/`

  This environment tries to balance a stick on a 2D moving car. Reward is based on the duration of "balancing". I also added a reward for the stability of the stick, i.e. penalizing moving away from the center and large velocity of the car and large angular velocity of the stick. This made training easier and quicker.



  The modified reward function is calculated as:

$$r_t = C \cdot \left(1 - \frac{|v_t|}{v_{\max}}\right) \cdot \left(1 - \frac{|\dot{\theta}_t|}{\dot{\theta}_{\max}}\right) \cdot \left(1 - \frac{|x_t|}{x_{\max}}\right) \quad (1)$$

  where:

  - r_t is the reward at time step t
  - C is a constant base reward (e.g., 1)
  - v_t is the cart's linear velocity
  - $v_{\max}$ is the maximum considered linear velocity
  - $\dot{\theta}_t$ is the angular velocity of the pole
  - $\dot{\theta}_{\max}$ is the maximum considered angular velocity
  - x_t is the cart's position
  - $x_{\max}$ is the maximum allowed position displacement from center

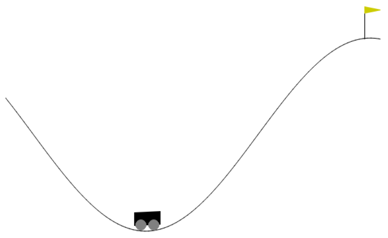
If the stick reaches large absolute value of the angle or large distance from the centre, the episode ends and a large negative reward (e.g. -100) is given.

The episode's maximum step count is set to a constant value (e.g. 100) and no additional reward is given at the end of a successful episode.

- `MountainCar`

  `https://gymnasium.farama.org/environments/classic_control/mountain_car/`

  I tried to also add other environments to my RL framework, such as Mountain Car, but I encountered difficulties with this environment. Additional fine-tuning of the Rust implementation would be required or building the environment from scratch in Rust to make this environment work and train well. Currently only QLearning/Sarsa produce some results after a long training period.



Results

Results were obtained by running each agent until convergence on the `CartPole` environment. Only runs where the reward function improved were considered. The average rewards were calculated over 10 runs for each agent.

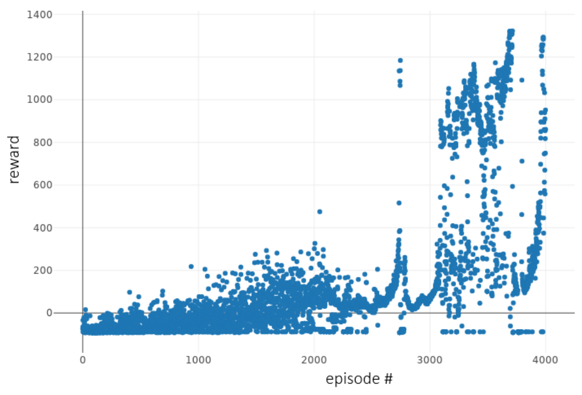
The reward at each timestep is calculated as described above. Since I'm using my own modified version of the cartpole environment, the numbers are arbitrary and should only be compared relative to each other.

Agent	**Avg Reward (10 runs)**
QLearning	1011.2
SARSA	953.8
MCTS	921.3
AlphaZero	1192.0
MuZero	923.0

Example Plots

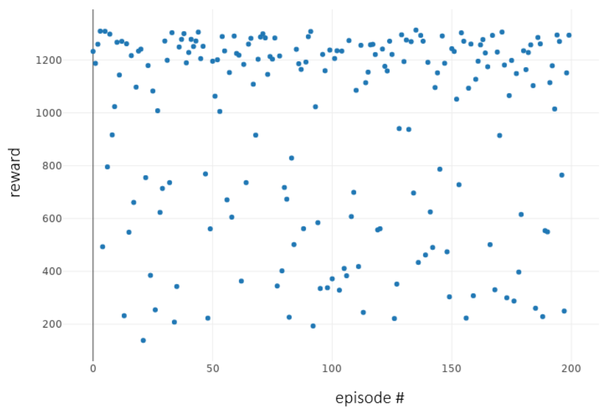
Plots below show performance during training. Each entry in the plot represents a total reward of the episode for a given episode number.

1. Qlearning / SARSA

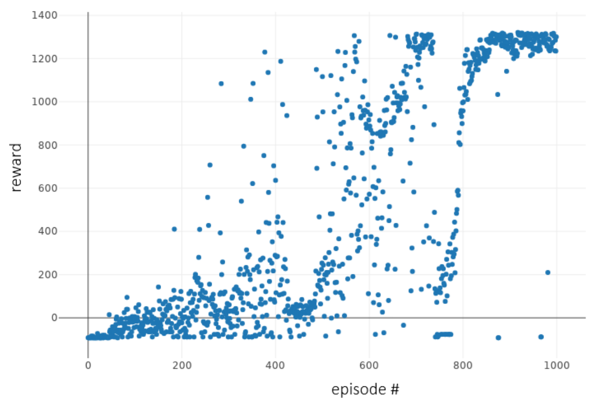


2. MCTS

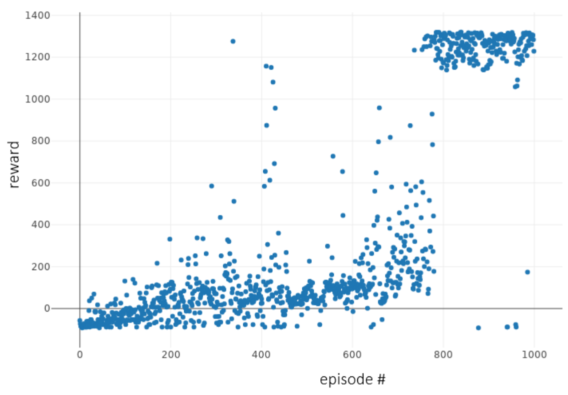
since MCTS is not a training algorithm, the plot below shows just a benchmark of runs, i.e. no improvement is expected over the increase in episode counts.



3. AlphaZero



4. MuZero



Installation and Startup Instructions

Prerequisites

- Rust (edition 2024 or newer)

Example Running the Application

```
cargo run –r –p mcrs2 –– sarsa ––episodes 2000
```

CLI Help

```
cargo run –r –p mcrs2 –– ––help
```