Reinforcement Learning-Based Agent Design

Implementation in Gymnasium Environments

Alessio Russo
Department of Mathematical, Physical and Computer Sciences
University of Parma

What is Reinforcement Learning?

Reinforcement learning is a learning paradigm based on trial-and-error interaction between an agent and an environment. In particular:

- An agent observes the environment state, selects an action a, and receives a scalar reward r.
- The environment transitions to a new state s in response to the action.
- The objective is to learn a policy that maximizes expected cumulative reward over time.
- Training data is generated online by the agent itself, not from a fixed dataset.
- Policies are commonly represented by neural networks trained using backpropagation.

Observations may be partial or noisy representations of the true environment state.

Key Challenges in Reinforcement Learning

Challenges specific to the reinforcement learning setting:

- Data is non-independent and non-identically distributed, since the agent's policy determines which experiences are collected.
- Exploration is required to discover rewarding actions, but excessive exploration degrades short-term performance.
- Rewards are often delayed, making it difficult to associate actions with their long-term consequences.

Gymnasium: The RL Environment Standard

Gymnasium specifies a common abstraction for reinforcement learning environments.

Environment abstraction:

- An environment is exposed through a unified interface.
- Interaction follows a well defined agent environment loop.
- This separation allows algorithms to be reused across tasks.

Core elements:

- Action space defines what decisions are available to the agent.
- Observation space defines how the environment state is perceived.
- `reset()` initializes an episode.
- `step(a)` advances the system by one interaction step.

Monte Carlo Tree Search (MCTS)

MCTS incrementally builds a partial search tree from the current state, using simulated trajectories to estimate action values and guide long-term decision making.

Four Phases (one MCTS iteration)

1. **Selection:** starting from the root, recursively select child nodes by balancing exploitation of high-value actions and exploration of less-visited ones.
2. **Expansion:** when a non-terminal node with unexplored actions is reached, expand the tree by generating a new child node.
3. **Simulation (Rollout):** from the newly expanded node, simulate the environment forward using a simple policy to estimate future reward.
4. **Backpropagation:** propagate the simulated return back through the visited nodes, updating visit counts and value estimates.

MCTS applied to CartPole

Task setting:

- Control a cart to keep an attached pole balanced in an upright position.
- Performance is measured by episode duration.

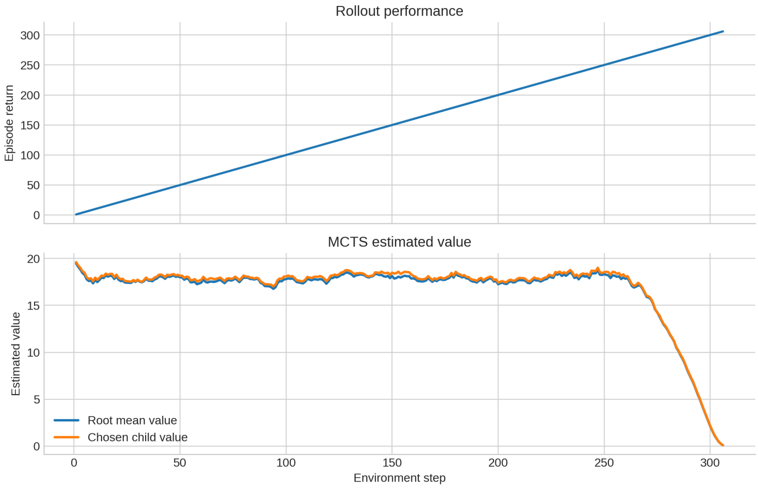
Environment description:

- Observations are low dimensional state vectors describing cart and pole dynamics.
- State variables include cart position and velocity, and pole angle and angular velocity.
- Actions correspond to discrete left or right forces applied to the cart.

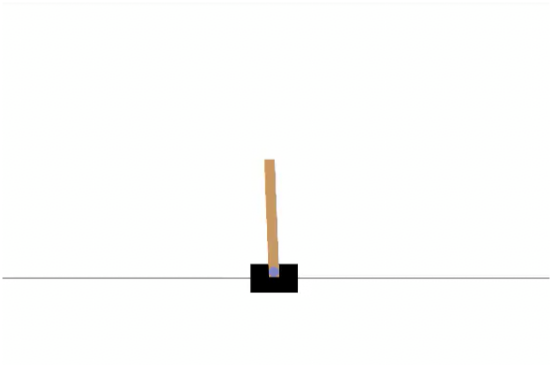
Reward design:

- The agent receives a constant positive reward at each time step.
- Reward accumulation directly reflects how long the pole remains balanced.
- Episodes terminate when the pole falls or the cart leaves the allowed range.

Results for CartPole



MCTS demonstration on CartPole



MCTS controlled agent interacting with the CartPole environment

Cross Entropy Method (CEM)

Core idea:

- Maintain a probability distribution over candidate solutions.
- Sample multiple solutions and evaluate them by running full episodes.
- Select the highest performing samples as elite.
- Update the distribution to increase the likelihood of elite solutions.

Use in reinforcement learning:

- The policy is represented by a set of parameters.
- Policies are sampled, executed in the environment, and scored by total reward.
- Only top performing policies influence the next parameter distribution.
- Repeating this process progressively concentrates on better policies.

CEM applied to LunarLander

Task setting:

- Control a spacecraft to achieve a safe and fuel efficient landing.
- Performance is evaluated over complete episodes.

Environment description:

- Observations encode position, velocity, orientation, and ground contact.
- Actions correspond to discrete engine commands controlling thrust.

Reward design:

- Continuous shaping encourages stable descent and correct positioning.
- Fuel usage is penalized to discourage unnecessary thrust.
- Successful landings are strongly rewarded, crashes are heavily penalized.

CEM configuration for LunarLander

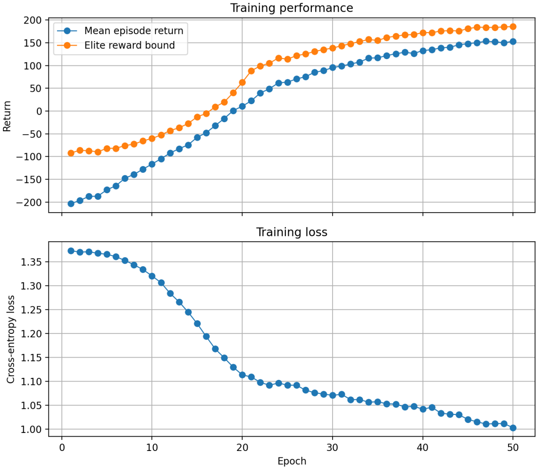
Model architecture:

- Input: low-dimensional state vector (flattened environment observation).
- Feature mapping: compact MLP with ReLU nonlinearities to capture state interactions.
- Policy head: final linear layer producing action logits for discrete control.
- Output: categorical action distribution over the 4 LunarLander actions.

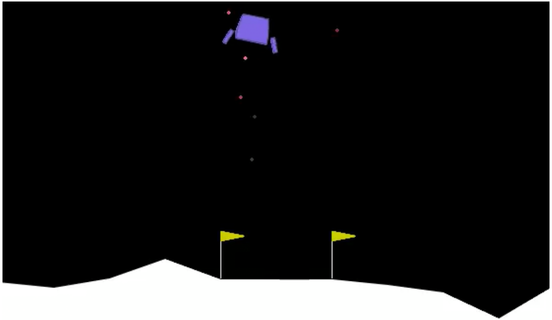
Training and sampling:

- Adam optimizer with learning rate 10^{-3} and cross-entropy loss.
- 1000 episodes sampled per epoch; top 20% retained by return.
- Episodes truncated at 300 steps; 10 optimization steps per epoch.

Training results for LunarLander



CEM demonstration on LunarLander



CEM controlled agent interacting with the LunarLander environment

CEM applied to Galaxian

Task setting:

- Control a spacecraft to survive enemy waves and score points.
- Performance is evaluated over complete game episodes.

Environment description:

- Observations are raw RGB images representing the full game screen.
- Visual input encodes player position, enemies, bullets, and background.
- Actions correspond to discrete controls such as moving left, moving right, and firing.

Reward design:

- Positive reward is given for destroying enemies.
- Survival is encouraged by avoiding terminal states.
- Episodes terminate when the player ship is destroyed.

CEM configuration for Galaxian

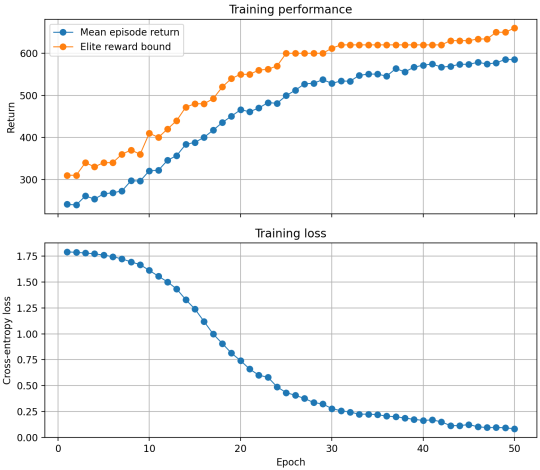
Model architecture:

- Input: single RGB frame resized to 96×96 and normalized per channel.
- Feature extractor: deep convolutional network capturing spatial structure of Atari frames.
- Policy head: fully connected layers mapping visual features to action logits.
- Output: categorical action distribution over the discrete Galaxian action space.

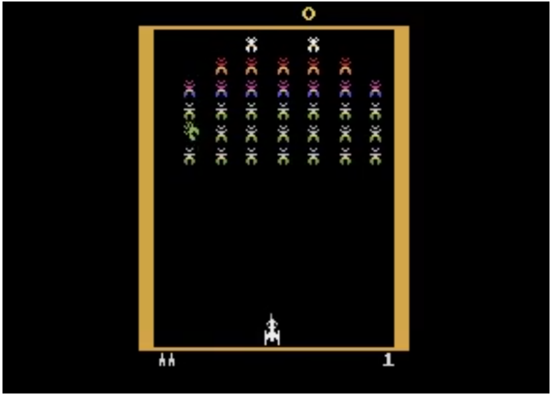
Training and sampling:

- Adam optimizer with learning rate 10^{-3} and cross-entropy loss.
- 300 episodes sampled per epoch; top 20% retained by return.
- Episodes truncated at 400 steps; 20 optimization steps per epoch.

Training results for Galaxian



CEM demonstration on Galaxian



CEM controlled agent interacting with the Galaxian environment

Thank you for your attention!

Alessio Russo
University of Parma

Code: `https://github.com/blkdmr/rl-algorithms`