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?

Learning setup and objective:

- Reinforcement learning is a learning paradigm based on trial-and-error interaction with an environment.
- An agent observes states, selects actions, and receives scalar reward signals.
- The objective is to learn a policy that maximizes expected cumulative reward over time.
- Training data is generated online by the agent itself rather than from a fixed dataset.
- Learning is commonly implemented with deep neural networks optimized using backpropagation.

Key Challenges in Reinforcement Learning

Fundamental difficulties:

- Data is non independent and non identically distributed because the agent controls the data collection process through its policy.
- Exploration is required to discover rewarding actions, but excessive exploration reduces short term performance.
- Rewards are often delayed, obscuring the relationship between actions and their long term consequences.

Agent-Environment Interaction

Reinforcement learning is defined by a repeated interaction loop between two components.

Entities involved:

- Agent: the decision making system that selects actions.
- Environment: the external system that responds to actions.

Information exchanged at each step:

- Action (a): a control signal chosen by the agent, discrete or continuous.
- State (s): the underlying configuration of the environment.
- Reward (r): a scalar signal evaluating the action outcome.

Note: an observation is a partial or noisy view of the true environment state.

Markov Process (MP)

A Markov Process (MP) is a mathematical model for systems that evolve randomly over time.

Markov property:

$$P(s_{t+1} \mid s_t) = P(s_{t+1} \mid s_1, s_2, \ldots, s_t)$$

Interpretation:

- The current state is a complete summary of the past.
- Knowing earlier states provides no additional predictive power.
- The system evolves through probabilistic state transitions.
- These transitions are specified by a transition matrix T, where

$$T_{i,j} = P(S_{t+1} = j \mid S_t = i)$$

Markov Reward Process (MRP)

A Markov Reward Process (MRP) builds on a MP by introducing a notion of desirability.

Interpretation:

- Each state or transition produces a scalar reward signal.
- Rewards define what outcomes are considered good or bad.
- Future rewards are discounted by a factor γ.
- The value function summarizes long term expected reward.

$$V(s) = \mathbb{E}[G_t \mid S_t = s]$$

Markov Decision Process (MDP)

A Markov Decision Process (MDP) extends a MRP by introducing actions that influence system dynamics.

At each time step:

- The agent observes the current state.
- The agent selects an action.
- The environment transitions to a new state and emits a reward.

Transitions and rewards depend on the chosen action:

$$P(s' \mid s, a), \qquad R(s, a)$$

Policy

The policy specifies how the agent behaves in each state.

Stochastic policy definition:

$$\pi(a \mid s) = P(A_t = a \mid S_t = s)$$

Interpretation:

- The policy maps states to probabilities over actions.
- Stochasticity allows exploration of alternative behaviors.
- Fixing the policy removes choice and yields a Markov Reward Process.

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

Monte Carlo Tree Search is a planning algorithm for sequential decision making.

Core idea:

- Decisions are evaluated by simulating possible future action sequences.
- A search tree is grown incrementally from the current state.
- Computational effort is focused on actions that appear promising.

Algorithm structure:

- The tree alternates between states and actions.
- Simulations estimate long term outcomes without requiring a learned model.
- Action quality is refined through repeated sampling.

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.

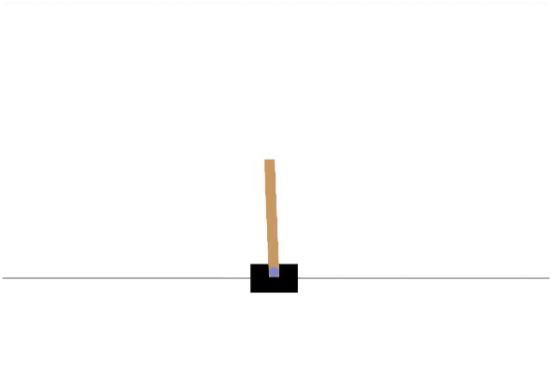
Environment description:

- 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.

MCTS demonstration on CartPole



MCTS controlled agent interacting with the CartPole environment

Cross Entropy Method

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

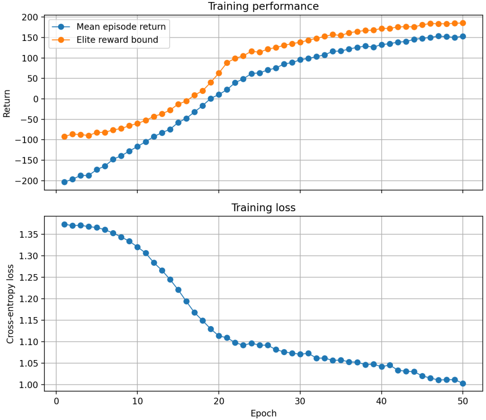
Model architecture:

- 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:

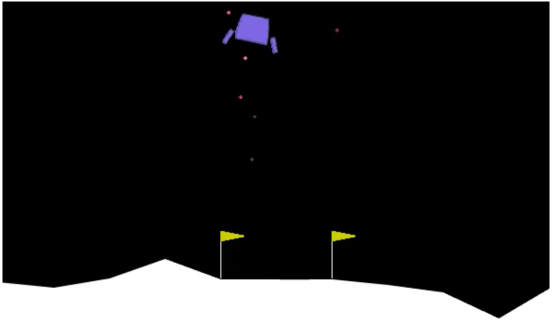
- Adam optimizer with learning rate 10^{-3} and cross-entropy loss.
- 1000 episodes sampled per epoch; top 20% retained by return.
- Episodes truncated at 300 steps; 10 optimization steps per epoch.

Training results for LunarLander



Training performance of CEM on the LunarLander environment

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

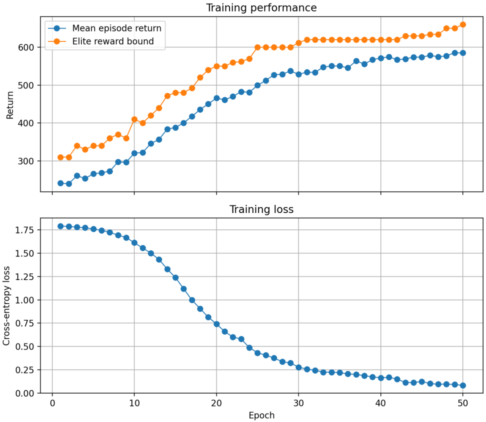
Model architecture:

- 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:

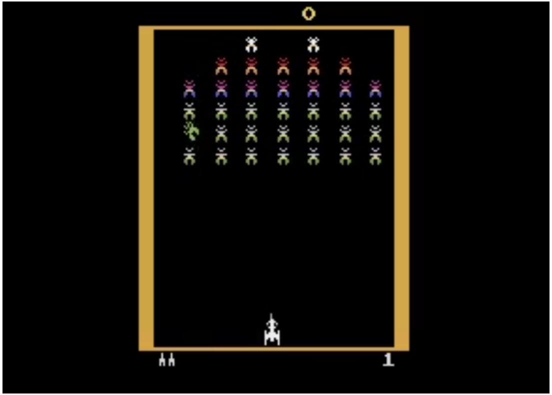
- Adam optimizer with learning rate 10^{-3} and cross-entropy loss.
- 300 episodes sampled per epoch; top 20% retained by return.
- Episodes truncated at 400 steps; 20 optimization steps per epoch.

Training results for Galaxian



Training performance of CEM on the Galaxian environment

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`