Reinforcement Learning Repository at UMass, Amherst

Demos and Implementation (Domains)

This section contains programs which demonstrate reinforcement learning in action, as an illustration of the concepts and common algorithms. These programs might provide a useful starting place for the implementation of reinforcement learning to solve real problems and advance research in this area. Wherever possible, source code is included.

Please note that use of this software is restricted; you must read this license agreement and agree to its terms before downloading any software from this site. Downloading the software is considered consent to the terms.

If you would like to contribute source code or make suggestions for improvement of what is included here, contact Bruno Castro da Silva or Sridhar Mahadevan.

- Cart-Pole Problem
Simulation of the cart and pole dynamic system and a procedure for learning to balance the pole. Both are described in Barto, Sutton, and Anderson, "*Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," IEEE Trans. Syst., Man, Cybern., Vol. SMC-13, pp. 834--846, Sept.--Oct. 1983* Written by Rich Sutton.
Source code: cpole.tar (16 K, requires C compiler)

- Cell Phone
Interactive Java demonstration illustrating the improvement gained by applying RL to the problem of Dynamic Channel Allocation in Cellular Telephones, by Satinder Singh at the University of Colorado

---

- Elevator
Fortran simulation of an elevator, written by James Lewis, and provided by Christos Cassandras at UMASS ECE Dept. The reinforcement learning addition to the elevator simulation was implemented by Bob Crites, CS Dept. UMass. and John McNulty
and is described in the paper Improving Elevator Performance Using Reinforcement Learning. Source code: elevator.tar.gz (284 K) or elevator.tar (814 K). Both require a C compiler and the f2c library to convert Fortran to c, as it incorporates c random number handling routines).

- Grid World This program is a simulation of learning the goal of moving to a user-defined square of a grid. It uses Q-learning, and was written by Sridhar Mahadevan.
Source code:grid.tar (72 K; requires C compiler and X11 libraries)

- Interactive Demo of Q-learning
A Java swing applet that allows the user to construct a grid by specifying danger and target cells, and then modify various learning paramaeters. Upon completion of learning, the learned policy is represented as arrows overlaying the grid. Documentation is available here. A french version is available here with documentation. Written by Thierry Masson. Requirements: JDK 1.3 or higher. Source code: TM_QLearnerDemo_Src_only.jar (38.8 K). Classes: TM_QLearnerDemo.jar (22.6 K).

- Least-Squares Policy Iteration
MatLab implementation of Least-Squares Policy Iteration (LSPI) algorithm. Documentation and background is available here. Written by Michail G. Lagoudakis and Ronald Parr. Requirements: MATLAB V.5 or higher. Source code: lspi.tar.gz (10.9 K), chain.tar.gz (12.2 K), pendulum.tar.gz (26.1 K).

- Machine Maintenance
CSIM simulation of a production system which integrates SMART, a model-free average-reward algorithm, to determine the optimal machine maintenance policy. It was written by Nicholas Marchalleck and Abhijit Gosavi, and is described in Self-Improving Factory Simulation using Continuous-Time Average-Reward Reinforcement Learning by Mahadevan et. al.
Source code: maint.tar (268 K; requires CSIM v.17 and C++ compiler)

- MDP Q-learning: implements Q-learning on a given MDP, using semi-uniform exploration.
Source code: mdp-q.tar (64 K, requires GNU C compiler)

- Mountain-Car Problem:
Simulation of a car learning the proper acceleration to get up a mountain. It uses Q-learning with CMAC as a function approximator. It is described in (among other papers) Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding by Rich Sutton, and was developed by Sridhar Mahadevan.
Source code: mcar.tar (157K; requires X11 libraries and C++ compiler)

- Network Routing: Demonstrates a RL network-routing algorithm written by Justin Boyan and Michael Littman. Described in Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach
Advances in Neural Information Processing Systems (Postscript - 155KB)
Source code: network-router.tar (222 K); requires C compiler, wish windowing shell (part of Tcl)

- Proposed Standard for Reinforcement Learning Software This standard, developed by Rich Sutton and Juan Carlos Santamaria, is intended to facilitate RL research and development, and is available for C++ and Common Lisp.

- Proposed Standard Interface
A proposed standard interface for RL systems written in C++. It provides standard interface classes for an agent, an environment, a function approximator, and states and actions. Written by Bohdana Ratitch.
Source code: si-classes.tar (72 K, requires C++ compiler, documentation).

Programs written by Bohdana Ratitch using this proposed standard interface are the following. All require a C++ compiler. Further compiler information can be found here.

- Random MDP generator. Source code: rmdp.tar (205 K, documentation).
- Mountain-Car task implementation. Source code: mc.tar (82 K, documentation).
- Randomized Mountain-Car task implementation. Source code: mcr.tar (82 K, documentation).
- SARSA(lambda) with replacing eligibility traces. Source code: sarsa.tar (82 K, documentation).
- CMAC function approximator. Source code: cmac.tar (82 K, documentation).
- Randomized Mountain-Car task using SARSA(lambda) and CMAC function approximator. Source code: mcr-sarsa-cmac.tar (154 K, documentation).
- Random MDP generator with SARSA(lambda) and CMAC function approximator. Source code: rmdp-sarsa-cmac.tar (348 K, documentation).

- Reinforcement Learning Toolbox

The Reinforcement Learning Toolbox is a set of classes implementing a variety of reinforcement learning algorithms, including TD-lambda, actor critic, prioritzed sweeping, and hierarchical learning. The toolbox also permits logging and error recognition. Written by Gerhard Neumann and Stephan Neumann. Download (written in C++, available for both Windows and Linux).

- Robot-on-a-grid Demo

A demonstration of the robot-on-a grid problem. Different parameters can be modified by the user, such as the selection strategy and learning method. Written by Gilad Mishne. Demo: ML2 project app.zip (2478 K, Windows executable), Source code: ML2 project src.zip (19 K), Sample grids: ML2 project grids.zip (1.3 K).

- Rumpus Gridworld Simulator

A language independent simulator that uses TCP/IP ports for interaction with client applications (agents). Requires the scripting language Ruby. Simulator allows a gridworld to be specified using a bitmap format and supports both local and unique state descriptions as well as deterministic and non-deterministic actions. Written by Torbjorn Dahl.

- CLSquare

CLSquare from the Neuroinformatics Group at the University of Osnabrueck simulates a control loop for closed loop control. Although originally designed for training and testing Reinforcement Learning controllers, it also applies to other learning and non-learning controller concepts. Currently availabe plants: Acrobot, bicycle, cart pole, cart double pole, pole, mountain car and maze. Currently availabe controllers: linear controller, Reinforcement learning Q table, neural network based Q controller. It comes with many useful features, e.g. graphical display and statistics output, a documentation, and many demos for quick starting.

- PIQLE

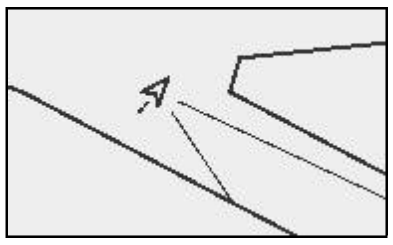
PIQLE is an open source set of Java classes for quickly experimenting with single- and multi-agent reinforcement learning schemes (new problems or new algorithms) by Francesco De Comité. Version 2, with with a major refactoring of classes, English renamings and synthetic documentation released in November 2006.

- Connectionist Q-learning Java Framework

The Free Connectionist Q-learning Java Framework is an open source Java library for developing learning systems using reinforcement learning and neural networks, by Dominik Kapusta.

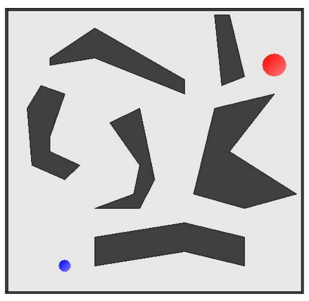
- Neuropilot Demos

Michael Fairbank's demonstrations of RL and RNNs learning in the Neuropilot Domain, including value-gradient learning.



- The Pinball Domain

The Pinball domain is a fairly challenging 4-dimensional continuous and dynamic reinforcement learning domain. The goal is to maneuver the blue ball into the red hole, while avoiding (or using, since the ball is dynamic and collisions are elastic) the obstacles. The dynamics of the ball and the presence of obstacles result in a domain with sharp discontinuities, and the location and shape of the obstacles can be specified so you can make the domain as hard or as easy as you want.



The source code is in Java, and includes full documentation, an RL-Glue interface, and GUI programs for editing obstacle configurations, viewing saved trajectories, etc.
