Reinforcement Learning Toolbox™ Release Notes

MATLAB®

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
☎ Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Reinforcement Learning Toolbox™ Release Notes

Trademarks

Patents

Contents

R2019a

1

R2019a

Version: 1.0

New Features

Reinforcement Learning Algorithms: Train policies using DQN, DDPG, A2C, and other algorithms

Using Reinforcement Learning Toolbox™ software, you can train policies using several standard reinforcement learning algorithms. You can create agents to train policies for the following:

- Q-learning
- SARSA
- Deep Q-networks (DQN)
- Deep deterministic policy gradients (DDPG)
- Policy gradient (PG)
- Advantage actor-critic (A2C)

You can also train policies using other algorithms by creating a custom agent.

For more information on creating and training agents, see “Reinforcement Learning Agents” and “Train Reinforcement Learning Agents”.

Environment Modeling: Create MATLAB and Simulink environment models and provide observation and reward signals for training policies

In a reinforcement learning scenario, the environment models the dynamics and system behavior with which the agent interacts. To define an environment model, you specify the following:

- Action and observation signals that the agent uses to interact with the environment.
- Reward signal that the agent uses to measure its success.
- Environment dynamic behavior.

You can model your environment using MATLAB® and Simulink®. For more information, see “Create MATLAB Environments for Reinforcement Learning” and “Create Simulink Environments for Reinforcement Learning”

Policy and Value Function Representation: Parameterize policies using deep neural networks, linear basis functions, and look-up tables

Reinforcement Learning Toolbox software provides objects for actor and critic representations. The actor represents the policy that selects the action to take. The critic represents the value function that estimates the value of the current policy. Depending on your application and selected agent, you can define policy and value functions using deep neural networks, linear basis functions, or look-up tables. For more information, see "Create Policy and Value Function Representations".

Interoperability: Import policies from Keras and the ONNX model format

You can import existing deep neural network policies and value functions from other deep learning frameworks, such as Keras and the ONNX™ format. For more information, see "Import Policy and Value Function Representations".

Training Acceleration: Parallelize environment simulations and gradient calculations on GPUs and multicore CPUs for policy training

You can accelerate policy training by running parallel training simulations. If you have:

- Parallel Computing Toolbox™ software, you can run parallel simulations on multicore computers
- MATLAB Parallel Server™software, you can run parallel simulations on computer clusters or cloud resources

You can also speed up deep neural network training and inference with high-performance NVIDIA® GPUs.

For more information, see "Train Reinforcement Learning Agents".

Code Generation: Deploy trained policies to embedded devices through automatic code generation for CPUs and GPUs

Once you have trained your reinforcement learning policy, you can generate code for policy deployment. You can generate optimized CUDA® code using GPU Coder™ and C/C ++ code using MATLAB Coder™.

You can deploy trained policies as C/C++ shared libraries, Microsoft® .NET Frameworkassemblies, Java® classes, and Python® packages.

For more information, see “Deploy Trained Reinforcement Learning Policies”.

Reference Examples: Implement controllers using reinforcement learning for automated driving and robotics applications

This release includes the following examples on training reinforcement learning policies for robotics and automated driving applications:

- “Train DDPG Agent to Control Flying Robot”
- “Train Biped Robot to Walk Using DDPG Agent”
- “Train DQN Agent for Lane Keeping Assist”
- “Train DDPG Agent for Adaptive Cruise Control”
- “Train DDPG Agent for Path Following Control”