Reinforcement Learning Algorithms

Assumptions

1. Python Docstring Format

We will utilize the **reStructuredText (reST)** format for Python docstrings. This format is both easy to read and write, and is widely supported by various documentation generation tools.

2. Executable Files

By "executable" files, we mean that each Python file should be runnable as a script from the command line. This is achieved by:

- Including a shebang line at the top of each file (`#!/usr/bin/env python3`).
- Defining a `main` function in each file that executes the relevant code when the file is run as a script.

3. Algorithm Function Parameters and Returns

We assume that the parameters for each function include:

- The environment (`env`)
- A value function or action-value function (`V` or `Q`)
- A policy
- Various parameters controlling the algorithm's behavior (e.g., `number of episodes`, `maximum steps per episode`, `learning rate alpha`, `discount factor gamma`, `trace decay parameter lambda`, etc.)

The functions are expected to return the updated value function or action-value function after the learning process.

Core Classes, Functions, and Methods

1. `monte_carlo.py`

This file will contain the function for the **Monte Carlo algorithm**. The function will accept the parameters described above, execute the Monte Carlo algorithm to update the value function based on the policy, and return the updated value function.

2. `td_lambtha.py`

This file will house the function for the **TD(λ) algorithm**. Similar to the Monte Carlo function, it will take the parameters outlined above, perform the TD(λ) algorithm to update the value function according to the policy, and return the updated value function.

3. `sarsa_lambtha.py`

This file will contain the function for the **SARSA(λ) algorithm**. The function will also take the parameters described above, execute the SARSA(λ) algorithm to update the action-value function based on the policy, and return the updated action-value function.

Resources

Read or Watch

- RL Course by David Silver - Lecture 4: Model-Free Prediction
- RL Course by David Silver - Lecture 5: Model Free Control
- Simple Reinforcement Learning: Temporal Difference Learning
- On-Policy TD Control

Definitions to Skim

- Monte Carlo method
- Temporal difference learning
- State–action–reward–state–action

Learning Objectives

- What is Monte Carlo?
- What is Temporal Difference?
- What is bootstrapping?
- What is n-step temporal difference?
- What is TD(λ)?
- What is an eligibility trace?
- What is SARSA? SARSA(λ)? SARSAMAX?
- What is 'on-policy' vs 'off-policy'?

Requirements

General

- Allowed editors: vi, vim, emacs
- All files will be interpreted/compiled on Ubuntu 16.04 LTS using python3 (version 3.5)
- Files will be executed with numpy (version 1.15), and gym (version 0.7)
- All files should end with a new line
- The first line of all files should be exactly `#!/usr/bin/env python3`
- A README.md file, at the root of the folder of the project, is mandatory
- Code should use the pycodestyle style (version 2.4)
- All modules, classes, and functions must have documentation

Monte Carlo Method: A Brief Overview

Monte Carlo methods are computational algorithms that use repeated random sampling to obtain numerical results. They are particularly useful in solving problems that are deterministic in nature but are

complex to solve directly. These methods find applications in various fields like physics, business risk calculation, and systems engineering.

The Monte Carlo method involves defining a domain of possible inputs, generating random inputs from this domain, performing deterministic computations on these inputs, and then aggregating the results. For example, it can be used to approximate the value of π by randomly placing points in a square and calculating the ratio of points that fall within an inscribed quadrant.

Despite its simplicity, the Monte Carlo method can be computationally expensive and may require parallel computing strategies for more complex problems.

Read More on Wikipedia