reinforce_mu_project

Or Fadida

Jul 30, 2025

CONTENTS:

sphinx-quickstart on Tue Jul 29 20:41:19 2025. You can adapt this file completely to your liking, but it should at least contain the root *toctree* directive.

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

REINFORCE

1.1 reinforce package

1.1.1 Submodules

1.1.2 reinforce.plotting module

This module defines all visualization functions used in the reinforcement learning experiments. It includes tools to plot reward trajectories, success rates, and convergence behavior across different learning rates and repetitions.

`reinforce.plotting.`**`create_eta_figure`**`(`*mu_runs*, *eta*, *step_indices*`)`

Creates a figure plotting: - 5 real runs based on rank positions (5th, 25th, 40th, 60th, 90th percentiles) - 1 stepwise median (50th percentile across all runs at each step), with proper math formatting.

Parameters

- **`mu_runs`** (*`NDArray[np.float64]`*) – Mu trajectories of shape (n_reps, n_steps).
- **`eta`** (*`float`*) – Learning rate value.
- **`step_indices`** (*`NDArray[np.int64]`*) – Step indices for x-axis.

Returns

The Matplotlib figure with six curves (5 real runs + median trajectory).

Return type

MPLFig

`reinforce.plotting.`**`create_reward_vs_y_figure`**`(`*reward_func*, *m*, *y_min*, *y_max*`)`

Creates a figure plotting the reward function against y values within a specified range.

Parameters

- **`reward_func`** (*`Callable[[float, float], float]`*) – Function to compute the reward given y and m.
- **`m`** (*`float`*) – Target value.
- **`y_min`** (*`float`*) – Minimum value of y for the plot.
- **`y_max`** (*`float`*) – Maximum value of y for the plot.

Returns

The Matplotlib figure and axes containing the plot.

Return type
Tuple[MPLFig, Axes]

`reinforce.plotting.`**`create_success_rate_vs_eta_figure`**(*etas*, *success_rates*)

Creates a figure plotting the convergence success rate against the learning rate (eta) and returns it.

Parameters

- **`etas`** (*`List[float]`*) – Learning rates (eta) for which success rates were computed.
- **`success_rates`** (*`NDArray[np.float64]`*) – Success rates corresponding to each eta.

Returns
The Matplotlib figure showing success rate vs. learning rate.

Return type
MPLFig

`reinforce.plotting.`**`figure_saving_assertion`**(*filename*, *directory_path_lst*, *extension*, *dpi*)

Asserts that the provided filename, directory path list, extension, and DPI are valid for saving a figure.

Parameters

- **`filename`** (*`str`*) – Base name for the file (without extension).
- **`directory_path_lst`** (*`List[str]`*) – List of folder names forming the relative path.
- **`extension`** (*`str`*) – File extension for saving the plot (e.g., 'png', 'pdf').
- **`dpi`** (*`int`*) – Resolution in dots per inch for saving the figure.

Return type
`None`

`reinforce.plotting.`**`save_figure`**(*fig*, *filename*, *directory_path_lst*, *extension*, *dpi*)

Saves a Matplotlib figure to a specified path with a unique filename.

Parameters

- **`fig`** (*`MPLFig`*) – The Matplotlib figure to save.
- **`filename`** (*`str`*) – Base name for the file (without extension).
- **`directory_path_lst`** (*`List[str]`*) – List of folder names forming the relative path.
- **`extension`** (*`str`*) – File extension for saving the plot (e.g., 'png', 'pdf').
- **`dpi`** (*`int`*) – Resolution in dots per inch for saving the figure.

Return type
`None`

1.1.3 reinforce.training module

This module implements training-related logic for updating the distribution parameter mu. It supports different reward structures and learning dynamics, and is designed to work with pluggable reward functions and gradients.

`reinforce.training.`**`run_multiple_repetitions`**(*m*, *mu0*, *n_steps*, *eta*, *n_reps*, *y_sampler*, *reward_func*, *g_func*, *g_tag_func*)

Runs multiple repetitions and returns an array of mu trajectories (n_reps, n_steps) for this eta value.

Parameters

- **`m`** (*`float`*) – Target value.
- **`mu0`** (*`float`*) – Initial mu value.
- **`n_steps`** (*`int`*) – Number of steps to run in each simulation.
- **`eta`** (*`float`*) – Learning rate.
- **`n_reps`** (*`int`*) – Number of repetitions to run.
- **`y_sampler`** (*`Callable[[float], float]`*) – Function to sample y given mu.
- **`reward_func`** (*`Callable[[float, float], float]`*) – Function to compute reward given y and m.
- **`g_func`** (*`Callable[[float, float], float]`*) – Function to compute g(y, mu).
- **`g_tag_func`** (*`Callable[[float, float], float]`*) – Function to compute the derivative of g with respect to mu.

Returns

Array of mu trajectories of shape (n_reps, n_steps).

Return type

NDArray[np.float64]

`reinforce.training.`**`run_single_run`**(*m*, *mu0*, *n_steps*, *eta*, *y_sampler*, *reward_func*, *g_func*, *g_tag_func*)

Runs a single REINFORCE simulation and returns the mu trajectory.

Parameters

- **`m`** (*`float`*) – Target value.
- **`mu0`** (*`float`*) – Initial mu value.
- **`n_steps`** (*`int`*) – Number of steps to run the simulation.
- **`eta`** (*`float`*) – Learning rate.
- **`y_sampler`** (*`Callable[[float], float]`*) – Function to sample y given mu.
- **`reward_func`** (*`Callable[[float, float], float]`*) – Function to compute reward given y and m.
- **`g_func`** (*`Callable[[float, float], float]`*) – Function to compute g(y, mu).

- **g_tag_func** (*Callable[[float, float], float]*) – Function to compute the derivative of g with respect to mu.

Returns
Array of mu values over the steps (trajectory).

Return type
NDArray[np.float64]

1.1.4 reinforce.utilities module

This module provides utility functions for the reinforcement learning setup. It includes tools for generating samples, defining reward functions, validating names, defining reward functions and organizing file paths.

`reinforce.utilities.`**`BASIC_SAFE_PATTERN = re.compile('^[A-Za-z0-9_\\- .]+$')`**

Regular expression pattern for validating basic filesystem names (alphanumeric, underscores, hyphens, spaces).

`reinforce.utilities.`**`REINFORCE_FUNCS`**

Type alias for a tuple of three functions used in the REINFORCE algorithm (y_sampler, g_func, g_tag_func).

alias of `Tuple[Callable[[float], float], Callable[[float, float], float], Callable[[float, float], float]]`

`reinforce.utilities.`**`asymmetric_reward_function`**(*y*, *m*)

Computes the asymmetric reward for a given value y and target m using squared distance.

Parameters

- **y** (*float*) – The sampled value.
- **m** (*float*) – The target value.

Returns
The reward value, two times the negative squared distance if y <= m, otherwise the negative squared distance.

Return type
float

`reinforce.utilities.`**`g_normal`**(*y*, *mu*, *sigma*)

Probability density of y given mu and sigma (normal distribution).

Parameters

- **y** (*float*) – The value for which to compute the density.
- **mu** (*float*) – Mean of the normal distribution.
- **sigma** (*float*) – Standard deviation of the normal distribution.

Returns
Probability density of y given mu and sigma.

Return type
float

`reinforce.utilities.`**`g_tag_normal`**`(`*y*, *mu*, *sigma*`)`

Calculates the derivative of g(y | mu, $sigma$) with respect to mu at a given y and returns it.

Parameters

- **y** (*float*) – The value for which to compute the gradient.
- **mu** (*float*) – Mean of the normal distribution.
- **sigma** (*float*) – Standard deviation of the normal distribution.

Returns

The derivative of the probability density function with respect to mu.

Return type

float

`reinforce.utilities.`**`generate_reinforce_normal_funcs`**`(`*sigma*`)`

Generates the REINFORCE functions for a normal distribution with given standard deviation. More specifically, it generates the sampling function for y, the probability density function g(y | mu, $sigma$), and the derivative of g with respect to mu.

Parameters

sigma (*float*) – Standard deviation of the normal distribution.

Returns

A tuple containing:

- y_sampler: Function to sample y given mu.
- g_func: Function to compute g(y | mu, $sigma$).
- g_tag_func: Function to compute the derivative of g with respect to mu.

Return type

REINFORCE_FUNCS

`reinforce.utilities.`**`get_unique_plot_filename`**`(`*base_name*, *extension='png'*`)`

Generates a unique filename by appending a number if the base name already exists.

Parameters

- **base_name** (*str*) – Base name for the file (relative or absolute path without extension).
- **extension** (*str*) – File extension (default is "png").

Returns

Unique filename with the specified base name and extension.

Return type

str

`reinforce.utilities.`**`is_valid_fs_name`**`(`*name*`)`

Checks if name is a valid filesystem name (not empty, alphanumeric, underscores, hyphens, and spaces).

Parameters

name (*str*) – Name to validate.

Returns
True if the name is valid, False otherwise.

Return type
bool

`reinforce.utilities.`**`is_valid_plt_extension`**`(`*extension*`)`

Checks if the given file extension is supported by Matplotlib for saving figures.

Parameters
`extension` (*`str`*) – File extension to check (e.g., ‘png’, ‘pdf’).

Returns
True if the extension is valid, False otherwise.

Return type
bool

`reinforce.utilities.`**`reward_function`**`(`*y*, *m*`)`

Computes the reward for a given value y and target m using squared distance.

Parameters

- **`y`** (*`float`*) – The sampled value.
- **`m`** (*`float`*) – The target value.

Returns
The reward value, negative squared distance.

Return type
float

`reinforce.utilities.`**`sample_y_normal`**`(`*mu*, *sigma*`)`

Samples a value y from a normal distribution centered at mu with standard deviation sigma.

Parameters

- **`mu`** (*`float`*) – Mean of the normal distribution.
- **`sigma`** (*`float`*) – Standard deviation of the normal distribution.

Returns
Sampled value y.

Return type
float

`reinforce.utilities.`**`scaled_reward_function`**`(`*y*, *m*`)`

Computes the scaled reward (scaled by 2) for a given value y and target m using squared distance.

Parameters

- **`y`** (*`float`*) – The sampled value.
- **`m`** (*`float`*) – The target value.

Returns
The reward value, two times the negative squared distance.

Return type
float

1.1.5 Module contents

The ‘reinforce’ package contains core components for the REINFORCE algorithm. It provides functionality for plotting results, training, parameter updates, and general-purpose utilities for managing trajectories and distributions. It is designed to facilitate the implementation and experimentation with reinforcement learning tasks.

CHAPTER

TWO

EXERCISE PACKAGE

2.1 Subpackages

2.1.1 exercise.scripts package

Submodules

exercise.scripts.assignment_constants module

This module defines global constants used across the reinforcement learning experiment. These constants include parameters for the experiment setup.

`exercise.scripts.assignment_constants.`**`dpi = 300`**

DPI (dots per inch) for saving plots, affecting the resolution of saved figures.

`exercise.scripts.assignment_constants.`**`etas_dict = {'part1': numpy.logspace.tolist, 'part3': [0.001]}`**

Dictionary mapping parts of the assignment to their respective etas (learning rates).

`exercise.scripts.assignment_constants.`**`extension = 'png'`**

File extension for saving plots.

`exercise.scripts.assignment_constants.`**`m = 2`**

The target value for the reward function.

`exercise.scripts.assignment_constants.`**`median_colors = ['red', 'green']`**

Colors for plotting final median values resulted from different sigma values on the reward vs. y plot.

`exercise.scripts.assignment_constants.`**`mu0 = 0`**

The initial mean for the normal distribution used to compute y values.

`exercise.scripts.assignment_constants.`**`n_reps = 200`**

Number of repetitions for each simulation to average results.

`exercise.scripts.assignment_constants.`**`n_steps = 10000`**

Number of steps to run in each REINFORCE simulation.

`exercise.scripts.assignment_constants.`**`reward_funcs_dict = {'part1': <function reward_function>, 'part3': <function asymmetric_reward_function>}`**

Dictionary mapping parts of the assignment to their respective reward functions.

`exercise.scripts.assignment_constants.`**`save_directories_dict = {'part1': ['exercise', 'plots', 'mu_trajectories'], 'part2': ['exercise', 'plots', 'success_rate_vs_eta'], 'part3': ['exercise', 'plots', 'reward_vs_y_with_median']}`**

Dictionary mapping parts of the assignment to their respective directory paths for saving plots.

`exercise.scripts.assignment_constants.`**`sigmas_dict = {'part1': [0.1], 'part3': [0.1, 0.5]}`**

Dictionary mapping parts of the assignment to their respective standard deviations.

`exercise.scripts.assignment_constants.`**`step_indices = 10000`**

Indices for each step in the simulation, used for plotting.

`exercise.scripts.assignment_constants.`**`tolerance = 0.1`**

Tolerance for checking convergence in the REINFORCE algorithm.

exercise.scripts.main module

Main script for running the $mu$-convergence reinforcement learning experiment.

This script configures the experiment using the constants file, initializes the learning loop, and produces relevant plots and statistics. Designed to be executed directly from the command line.

`exercise.scripts.main.`**`Float`**

Type alias for a generic float type.

alias of `float` | `floating`

`exercise.scripts.main.`**`compute_success_rate`**(*mu_trajectories*, *m*, *tolerance*)

Computes the success rate (percentage of runs) where the final mu converges within [m - tolerance, m + tolerance].

Parameters

- **`mu_trajectories`** (*`NDArray[np.float64]`*) – Array of mu trajectories with shape (n_reps, n_steps).
- **`m`** (*`float`*) – Target value.
- **`tolerance`** (*`float`*) – Acceptable deviation from target m for successful convergence.

Returns

Success rate as a fraction of runs that converged successfully.

Return type

float

`exercise.scripts.main.`**`compute_success_rates_from_trajectories`**(*eta_mu_pairs*, *m*, *tolerance*)

Computes success rates from precomputed mu trajectories per learning rate.

Parameters

- **`eta_mu_pairs`** (*`List[Tuple[float, NDArray[np.float64]]]`*) –

  List of tuples. Each tuple contains:

  – float : learning rate eta

  - np.ndarray : mu trajectories with shape (n_reps, n_steps).
- **m** (*float*) – Target value.
- **tolerance** (*float*) – Acceptable deviation from target m for successful convergence.

Returns
Array of success rates, ordered according to eta_mu_pairs.

Return type
NDArray[np.float64]

`exercise.scripts.main.`**`run_full_experiment`**(*reward_funcs_dict*, *m*, *tolerance*, *median_colors*, *mu0*, *sigmas_dict*, *n_steps*, *n_reps*, *etas_dict*, *step_indices*, *save_directories_dict*, *extension*, *dpi=300*)

Runs full experiment across different etas and saves the corresponding figures.

Parameters

- **reward_funcs_dict** (*Dict[str, Callable[[float, float], float]]*) – Dictionary with keys 'part1', 'part3', each containing a reward function.
- **m** (*float*) – Target value.
- **tolerance** (*float*) – Acceptable deviation from target m for successful convergence.
- **median_colors** (*List[str]*) – List of colors for plotting median points in part three.
- **mu0** (*float*) – Initial mu value.
- **sigmas_dict** (*Dict[str, List[float]]*) – Dictionary with keys 'part1', 'part3', each containing a list of floats representing the standard deviation for sampling y.
- **n_steps** (*int*) – Number of steps.
- **n_reps** (*int*) – Number of repetitions.
- **etas_dict** (*Dict[str, List[float]]*) – Dictionary with keys 'part1', 'part3', each containing a list of learning rates for different parts of the experiment.
- **step_indices** (*NDArray[np.int64]*) – Step indices for x-axis.
- **save_directories_dict** (*Dict[str, List[str]]*) – Dictionary with keys 'part1', 'part2', 'part3', each containing a list of folders forming the path to save plots.
- **extension** (*str*) – File extension for saving plots (e.g., 'png', 'pdf').
- **dpi** (*int*) – Dots per inch for saved plots (default is 300).

Return type
None

exercise.scripts.main.**run_part_one**(*y_sampler*, *reward_func*, *g_func*, *g_tag_func*, *m*, *mu0*, *n_steps*, *n_reps*, *etas*, *step_indices*, *save_directories*, *extension*, *dpi*)

Runs part one of the experiment: compute mu trajectories for different etas and save the figures.

Parameters

- **y_sampler** (*Callable[[float], float]*) – Function to sample y given mu.
- **reward_func** (*Callable[[float, float], float]*) – Function to compute reward given y and m.
- **g_func** (*Callable[[float, float], float]*) – Function to compute g(y, mu).
- **g_tag_func** (*Callable[[float, float], float]*) – Function to compute the derivative of g with respect to mu.
- **m** (*float*) – Target value.
- **mu0** (*float*) – Initial mu value.
- **n_steps** (*int*) – Number of steps.
- **n_reps** (*int*) – Number of repetitions.
- **etas** (*List[float]*) – List of learning rates.
- **step_indices** (*NDArray[np.int64]*) – Step indices for x-axis.
- **save_directories** (*List[str]*) – List of folders forming the path to save plots.
- **extension** (*str*) – File extension for saving plots (e.g., 'png', 'pdf').
- **dpi** (*int*) – Dots per inch for saved plots.

Returns
List of tuples containing eta and corresponding mu trajectories.

Return type
List[Tuple[float, NDArray[np.float64]]]

exercise.scripts.main.**run_part_three**(*y_sampler_lst*, *reward_func*, *g_func_lst*, *g_tag_func_lst*, *sigmas*, *median_colors*, *m*, *mu0*, *n_steps*, *n_reps*, *eta*, *save_directories*, *extension*, *dpi*)

Runs part three of the experiment: simulates REINFORCE with different sigmas and plots reward vs y with median points.

Parameters

- **y_sampler_lst** (*List[Callable[[float], float]]*) – List of functions to sample y given mu for different sigmas.
- **reward_func** (*Callable[[Float, Float], Float]*) – Function to compute reward given y and m.
- **g_func_lst** (*List[Callable[[float, float], float]]*) – List of functions to compute g(y, mu) for different sigmas.

- **g_tag_func_lst** (*List[Callable[[float, float], float]]*) – List of functions to compute the derivative of g with respect to mu for different sigmas.
- **sigmas** (*List[float]*) – List of standard deviations for sampling y.
- **median_colors** (*List[str]*) – List of colors for plotting median points.
- **m** (*float*) – Target value.
- **mu0** (*float*) – Initial mu value.
- **n_steps** (*int*) – Number of steps.
- **n_reps** (*int*) – Number of repetitions.
- **eta** (*float*) – Learning rate.
- **save_directories** (*List[str]*) – List of folders forming the path to save plots.
- **extension** (*str*) – File extension for saving plots (e.g., 'png', 'pdf').
- **dpi** (*int*) – Dots per inch for saved plots.

Return type

None

exercise.scripts.main.**run_part_two**(*eta_mu_pairs*, *m*, *tolerance*, *save_directories*, *extension*, *dpi*)

Runs part two of the experiment: computes success rates from mu trajectories and plot/save success rate vs eta.

Parameters

- **eta_mu_pairs** (*List[Tuple[float, NDArray[np.float64]]]*) – List of tuples containing eta and corresponding mu trajectories.
- **m** (*float*) – Target value.
- **tolerance** (*float*) – Acceptable deviation from target m for successful convergence.
- **save_directories** (*List[str]*) – List of folders forming the path to save the figure.
- **extension** (*str*) – File extension for saving the figure (e.g., 'png', 'pdf').
- **dpi** (*int*) – Dots per inch for saved plots.

Return type

None

Module contents

The scripts package contains runnable Python files for executing the main reinforcement learning experiments. These scripts use the core 'reinforce' package to configure and run different experimental setups.

2.2 Module contents

The ‘exercise’ package contains user-specific scripts and output directories for experiment execution. It is not part of the core package, but contains the runnable main script and static output data.

PYTHON MODULE INDEX

INDEX