RL Baselines3 Zoo Documentation

Release 2.4.0a0

Stable Baselines3 Contributors

May 03, 2024

USER GUIDE

RL Baselines3 Zoo s a training framework for Reinforcement Learning (RL), using Stable Baselines3 (SB3), reliable implementations of reinforcement learning algorithms in PyTorch.

Github repository: https://github.com/DLR-RM/rl-baselines3-zoo

It provides scripts for training, evaluating agents, tuning hyperparameters, plotting results and recording videos.

In addition, it includes a collection of tuned hyperparameters for common environments and RL algorithms, and agents trained with those settings.

CHAPTER

ONE

INSTALLATION

1.1 Prerequisites

RL Zoo requires python 3.8+ and PyTorch >= 1.13

1.2 Minimal Installation

To install RL Zoo with pip, execute:

```
pip install rl_zoo3
```

From source:

```
git clone https://github.com/DLR-RM/rl-baselines3-zoo
cd rl-baselines3-zoo/
pip install -e .
```

Note: You can do `python -m rl_zoo3.train` from any folder and you have access to `rl_zoo3` command line interface, for instance, `rl_zoo3 train` is equivalent to `python train.py`

1.3 Full installation

With extra envs and test dependencies:

Note: If you want to use Atari games, you will need to do `pip install "autorom[accept-rom-license]"` additionally to download the ROMs

```
      apt-get install swig cmake ffmpeg
      pip install -r requirements.txt
pip install -e .[plots,tests]
```

Please see Stable Baselines3 documentation for alternatives to install stable baselines3.

1.4 Docker Images

Build docker image (CPU):

```
make docker-cpu
```

GPU:

```
USE_GPU=True make docker-gpu
```

Pull built docker image (CPU):

```
docker pull stablebaselines/rl-baselines3-zoo-cpu
```

GPU image:

```
docker pull stablebaselines/rl-baselines3-zoo
```

Run script in the docker image:

```
./scripts/run_docker_cpu.sh python train.py --algo ppo --env CartPole-v1
```

CHAPTER

TWO

GETTING STARTED

Note: You can try the following examples online using Google Colab notebook: RL Baselines zoo notebook

The hyperparameters for each environment are defined in `hyperparameters/algo_name.yml`.

If the environment exists in this file, then you can train an agent using:

```
python -m rl_zoo3.train --algo algo_name --env env_id
```

Or if you are in the RL Zoo3 folder:

```
python train.py --algo algo_name --env env_id
```

For example (with evaluation and checkpoints):

```
python -m rl_zoo3.train --algo ppo --env CartPole-v1 --eval-freq 10000 --save-freq 50000
```

If the trained agent exists, then you can see it in action using:

```
python -m rl_zoo3.enjoy --algo algo_name --env env_id
```

For example, enjoy A2C on Breakout during 5000 timesteps:

```
python -m rl_zoo3.enjoy --algo a2c --env BreakoutNoFrameskip-v4 --folder rl-trained-
↪agents/ -n 5000
```

CHAPTER

THREE

TRAIN AN AGENT

3.1 Basic Usage

The hyperparameters for each environment are defined in `hyperparameters/algo_name.yml`.

Note: Once RL Zoo3 is install, you can do `python -m rl_zoo3.train` from any folder, it is equivalent to `python train.py`

If the environment exists in this file, then you can train an agent using:

```
python train.py --algo algo_name --env env_id
```

Note: You can use `-P` (`--progress`) option to display a progress bar.

3.2 Custom Config File

Using a custom config file when it is a yaml file with a which contains a `env_id` entry:

```
python train.py --algo algo_name --env env_id --conf-file my_yaml.yml
```

You can also use a python file that contains a dictionary called *hyperparams* with an entry for each `env_id`. (see `hyperparams/python/ppo_config_example.py` for an example)

```
# You can pass a path to a python file
python train.py --algo ppo --env MountainCarContinuous-v0 --conf-file hyperparams/python/
↪ppo_config_example.py
# Or pass a path to a file from a module (for instance my_package.my_file)
python train.py --algo ppo --env MountainCarContinuous-v0 --conf-file hyperparams.python.
↪ppo_config_example
```

The advantage of this approach is that you can specify arbitrary python dictionaries and ensure that all their dependencies are imported in the config file itself.

3.3 Tensorboard, Checkpoints, Evaluation

For example (with tensorboard support):

```
python train.py --algo ppo --env CartPole-v1 --tensorboard-log /tmp/stable-baselines/
```

Evaluate the agent every 10000 steps using 10 episodes for evaluation (using only one evaluation env):

```
python train.py --algo sac --env AntBulletEnv-v0 --eval-freq 10000 --eval-episodes 10 --
↪n-eval-envs 1
```

Save a checkpoint of the agent every 100000 steps:

```
python train.py --algo td3 --env AntBulletEnv-v0 --save-freq 100000
```

3.4 Resume Training

Continue training (here, load pretrained agent for Breakout and continue training for 5000 steps):

```
python train.py --algo a2c --env BreakoutNoFrameskip-v4 -i rl-trained-agents/a2c/
↪BreakoutNoFrameskip-v4_1/BreakoutNoFrameskip-v4.zip -n 5000
```

3.5 Save Replay Buffer

When using off-policy algorithms, you can also **save the replay buffer** after training:

```
python train.py --algo sac --env Pendulum-v1 --save-replay-buffer
```

It will be automatically loaded if present when continuing training.

3.6 Env keyword arguments

You can specify keyword arguments to pass to the env constructor in the command line, using `--env-kwargs`:

```
python enjoy.py --algo ppo --env MountainCar-v0 --env-kwargs goal_velocity:10
```

3.7 Overwrite hyperparameters

You can easily overwrite hyperparameters in the command line, using `--hyperparams`:

```
python train.py --algo a2c --env MountainCarContinuous-v0 --hyperparams learning_rate:0.
↪001 policy_kwargs:"dict(net_arch=[64, 64])"
```

Note: if you want to pass a string, you need to escape it like that: `my_string:"'value'"`

CHAPTER

FOUR

PLOT SCRIPTS

Plot scripts (to be documented, see "Results" sections in SB3 documentation):

- `scripts/all_plots.py/scripts/plot_from_file.py` for plotting evaluations
- `scripts/plot_train.py` for plotting training reward/success

4.1 Examples

Plot training success (y-axis) w.r.t. timesteps (x-axis) with a moving window of 500 episodes for all the `Fetch` environment with `HER` algorithm:

```
python scripts/plot_train.py -a her -e Fetch -y success -f rl-trained-agents/ -w 500 -x␣
↪steps
```

Plot evaluation reward curve for TQC, SAC and TD3 on the HalfCheetah and Ant PyBullet environments:

```
python3 scripts/all_plots.py -a sac td3 tqc --env HalfCheetahBullet AntBullet -f rl-
↪trained-agents/
```

4.2 Plot with the rliable library

The RL zoo integrates some of rliable library features. You can find a visual explanation of the tools used by rliable in this blog post.

First, you need to install rliable.

Note: Python 3.7+ is required in that case.

Then export your results to a file using the `all_plots.py` script (see above):

```
python scripts/all_plots.py -a sac td3 tqc --env Half Ant -f logs/ -o logs/offpolicy
```

You can now use the `plot_from_file.py` script with `--rliable`, `--versus` and `--iqm` arguments:

```
python scripts/plot_from_file.py -i logs/offpolicy.pkl --skip-timesteps --rliable --
↪versus -l SAC TD3 TQC
```

Note: you may need to edit `plot_from_file.py`, in particular the `env_key_to_env_id` dictionary and the `scripts/score_normalization.py` which stores min and max score for each environment.

Remark: plotting with the `--rliable` option is usually slow as confidence interval need to be computed using bootstrap sampling.

CHAPTER

FIVE

ENJOY A TRAINED AGENT

Note: To download the repo with the trained agents, you must use `git clone --recursive https://github.com/DLR-RM/rl-baselines3-zoo` in order to clone the submodule too.

5.1 Enjoy a trained agent

If the trained agent exists, then you can see it in action using:

```
python enjoy.py --algo algo_name --env env_id
```

For example, enjoy A2C on Breakout during 5000 timesteps:

```
python enjoy.py --algo a2c --env BreakoutNoFrameskip-v4 --folder rl-trained-agents/ -n␣
↪5000
```

If you have trained an agent yourself, you need to do:

```
# exp-id 0 corresponds to the last experiment, otherwise, you can specify another ID
python enjoy.py --algo algo_name --env env_id -f logs/ --exp-id 0
```

5.2 Load Checkpoints, Best Model

To load the best model (when using evaluation environment):

```
python enjoy.py --algo algo_name --env env_id -f logs/ --exp-id 1 --load-best
```

To load a checkpoint (here the checkpoint name is `rl_model_10000_steps.zip`):

```
python enjoy.py --algo algo_name --env env_id -f logs/ --exp-id 1 --load-checkpoint 10000
```

To load the latest checkpoint:

```
python enjoy.py --algo algo_name --env env_id -f logs/ --exp-id 1 --load-last-checkpoint
```

5.3 Record a Video of a Trained Agent

Record 1000 steps with the latest saved model:

```
python -m rl_zoo3.record_video --algo ppo --env BipedalWalkerHardcore-v3 -n 1000
```

Use the best saved model instead:

```
python -m rl_zoo3.record_video --algo ppo --env BipedalWalkerHardcore-v3 -n 1000 --load-
↪best
```

Record a video of a checkpoint saved during training (here the checkpoint name is `rl_model_10000_steps.zip`):

```
python -m rl_zoo3.record_video --algo ppo --env BipedalWalkerHardcore-v3 -n 1000 --load-
↪checkpoint 10000
```

5.4 Record a Video of a Training Experiment

Apart from recording videos of specific saved models, it is also possible to record a video of a training experiment where checkpoints have been saved.

Record 1000 steps for each checkpoint, latest and best saved models:

```
python -m rl_zoo3.record_training --algo ppo --env CartPole-v1 -n 1000 -f logs --
↪deterministic
```

The previous command will create a `mp4` file. To convert this file to `gif` format as well:

```
python -m rl_zoo3.record_training --algo ppo --env CartPole-v1 -n 1000 -f logs --
↪deterministic --gif
```

CHAPTER

SIX

CUSTOM ENVIRONMENT

The easiest way to add support for a custom environment is to edit `rl_zoo3/import_envs.py` and register your environment here. Then, you need to add a section for it in the hyperparameters file (`hyperparams/algo.yml` or a custom yaml file that you can specify using `--conf-file` argument).

CHAPTER

SEVEN

CONFIGURATION

7.1 Hyperparameter yaml syntax

The syntax used in `hyperparameters/algo_name.yml` for setting hyperparameters (likewise the syntax to overwrite hyperparameters on the cli) may be specialized if the argument is a function. See examples in the `hyperparameters/` directory. For example:

- Specify a linear schedule for the learning rate:

```
learning_rate: lin_0.012486195510232303
```

Specify a different activation function for the network:

```
policy_kwargs: "dict(activation_fn=nn.ReLU)"
```

For a custom policy:

```
policy: my_package.MyCustomPolicy  # for instance stable_baselines3.ppo.MlpPolicy
```

7.2 Env Normalization

In the hyperparameter file, `normalize:` `True` means that the training environment will be wrapped in a VecNormalize wrapper.

Normalization uses the default parameters of `VecNormalize`, with the exception of `gamma` which is set to match that of the agent. This can be overridden using the appropriate `hyperparameters/algo_name.yml`, e.g.

```
normalize: "{'norm_obs': True, 'norm_reward': False}"
```

7.3 Env Wrappers

You can specify in the hyperparameter config one or more wrapper to use around the environment:

for one wrapper:

```
env_wrapper: gym_minigrid.wrappers.FlatObsWrapper
```

for multiple, specify a list:

```
env_wrapper:
    - rl_zoo3.wrappers.TruncatedOnSuccessWrapper:
        reward_offset: 1.0
    - sb3_contrib.common.wrappers.TimeFeatureWrapper
```

Note that you can easily specify parameters too.

By default, the environment is wrapped with a `Monitor` wrapper to record episode statistics. You can specify arguments to it using `monitor_kwargs` parameter to log additional data. That data *must* be present in the info dictionary at the last step of each episode.

For instance, for recording success with goal envs (e.g. `FetchReach-v1`):

```
monitor_kwargs: dict(info_keywords=('is_success',))
```

or recording final x position with `Ant-v3`:

```
monitor_kwargs: dict(info_keywords=('x_position',))
```

Note: for known `GoalEnv` like `FetchReach`, `info_keywords=('is_success',)` is actually the default.

7.4 VecEnvWrapper

You can specify which `VecEnvWrapper` to use in the config, the same way as for env wrappers (see above), using the `vec_env_wrapper` key:

For instance:

```
vec_env_wrapper: stable_baselines3.common.vec_env.VecMonitor
```

Note: `VecNormalize` is supported separately using `normalize` keyword, and `VecFrameStack` has a dedicated keyword `frame_stack`.

7.5 Callbacks

Following the same syntax as env wrappers, you can also add custom callbacks to use during training.

```
callback:
  - rl_zoo3.callbacks.ParallelTrainCallback:
      gradient_steps: 256
```

CHAPTER

EIGHT

INTEGRATIONS

8.1 Huggingface Hub Integration

List and videos of trained agents can be found on our Huggingface page: https://huggingface.co/sb3

Upload model to hub (same syntax as for `enjoy.py`):

```
python -m rl_zoo3.push_to_hub --algo ppo --env CartPole-v1 -f logs/ -orga sb3 -m
↪"Initial commit"
```

you can choose custom `repo-name` (default: `{algo}-{env_id}`) by passing a `--repo-name` argument.

Download model from hub:

```
python -m rl_zoo3.load_from_hub --algo ppo --env CartPole-v1 -f logs/ -orga sb3
```

8.2 Experiment tracking

We support tracking experiment data such as learning curves and hyperparameters via Weights and Biases.

The following command

```
python train.py --algo ppo --env CartPole-v1 --track --wandb-project-name sb3
```

yields a tracked experiment at this URL.

To add a tag to the run, (e.g. `optimized`), use the argument `--wandb-tags optimized`.

CHAPTER

NINE

HYPERPARAMETER TUNING

9.1 Hyperparameter Tuning

We use Optuna for optimizing the hyperparameters. Not all hyperparameters are tuned, and tuning enforces certain default hyperparameter settings that may be different from the official defaults. See rl_zoo3/hyperparams_opt.py for the current settings for each agent.

Hyperparameters not specified in rl_zoo3/hyperparams_opt.py are taken from the associated YAML file and fallback to the default values of SB3 if not present.

Note: when using SuccessiveHalvingPruner ("halving"), you must specify `--n-jobs > 1`

Budget of 1000 trials with a maximum of 50000 steps:

```
python train.py --algo ppo --env MountainCar-v0 -n 50000 -optimize --n-trials 1000 --n-
↪jobs 2 \
  --sampler tpe --pruner median
```

Distributed optimization using a shared database is also possible (see the corresponding Optuna documentation):

```
python train.py --algo ppo --env MountainCar-v0 -optimize --study-name test --storage␣
↪sqlite:///example.db
```

Print and save best hyperparameters of an Optuna study:

```
python scripts/parse_study.py -i path/to/study.pkl --print-n-best-trials 10 --save-n-
↪best-hyperparameters 10
```

The default budget for hyperparameter tuning is 500 trials and there is one intermediate evaluation for pruning/early stopping per 100k time steps.

9.1.1 Hyperparameters search space

Note that the default hyperparameters used in the zoo when tuning are not always the same as the defaults provided in stable-baselines3. Consult the latest source code to be sure of these settings. For example:

- PPO tuning assumes a network architecture with `ortho_init = False` when tuning, though it is `True` by default. You can change that by updating rl_zoo3/hyperparams_opt.py.
- Non-episodic rollout in TD3 and DDPG assumes `gradient_steps = train_freq` and so tunes only `train_freq` to reduce the search space.

When working with continuous actions, we recommend to enable gSDE by uncommenting lines in rl_zoo3/hyperparams_opt.py.

CHAPTER

TEN

STABLE BASELINES JAX (SBX)

Stable Baselines Jax (SBX) is a proof of concept version of Stable-Baselines3 in Jax.

It provides a minimal number of features compared to SB3 but can be much faster (up to 20x times!): https://twitter.com/araffin2/status/1590714558628253698

It is also compatible with the RL Zoo. For that you will need to create two files.

`train_sbx.py`:

```
import rl_zoo3
import rl_zoo3.train
from rl_zoo3.train import train
from sbx import DQN, PPO, SAC, TQC, DroQ


rl_zoo3.ALGOS["tqc"] = TQC
rl_zoo3.ALGOS["droq"] = DroQ
rl_zoo3.ALGOS["sac"] = SAC
rl_zoo3.ALGOS["ppo"] = PPO
rl_zoo3.ALGOS["dqn"] = DQN
rl_zoo3.train.ALGOS = rl_zoo3.ALGOS
rl_zoo3.exp_manager.ALGOS = rl_zoo3.ALGOS

if __name__ == "__main__":
    train()
```

Then you can call `python train_sbx.py --algo sac --env Pendulum-v1` and use the RL Zoo CLI.

`enjoy_sbx.py`:

```
import rl_zoo3
import rl_zoo3.enjoy
from rl_zoo3.enjoy import enjoy
from sbx import DQN, PPO, SAC, TQC, DroQ


rl_zoo3.ALGOS["tqc"] = TQC
rl_zoo3.ALGOS["droq"] = DroQ
rl_zoo3.ALGOS["sac"] = SAC
rl_zoo3.ALGOS["ppo"] = PPO
rl_zoo3.ALGOS["dqn"] = DQN
rl_zoo3.enjoy.ALGOS = rl_zoo3.ALGOS
rl_zoo3.exp_manager.ALGOS = rl_zoo3.ALGOS
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    enjoy()
```

CHAPTER

ELEVEN

EXPERIMENT MANAGER

11.1 Parameters

class rl_zoo3.exp_manager.**ExperimentManager**(*args*, *algo*, *env_id*, *log_folder*, *tensorboard_log=''*, *n_timesteps=0*, *eval_freq=10000*, *n_eval_episodes=5*, *save_freq=-1*, *hyperparams=None*, *env_kwargs=None*, *eval_env_kwargs=None*, *trained_agent=''*, *optimize_hyperparameters=False*, *storage=None*, *study_name=None*, *n_trials=1*, *max_total_trials=None*, *n_jobs=1*, *sampler='tpe'*, *pruner='median'*, *optimization_log_path=None*, *n_startup_trials=0*, *n_evaluations=1*, *truncate_last_trajectory=False*, *uuid_str=''*, *seed=0*, *log_interval=0*, *save_replay_buffer=False*, *verbose=1*, *vec_env_type='dummy'*, *n_eval_envs=1*, *no_optim_plots=False*, *device='auto'*, *config=None*, *show_progress=False*)

Experiment manager: read the hyperparameters, preprocess them, create the environment and the RL model.

Please take a look at *train.py* to have the details for each argument.

Parameters

- **args** (*Namespace*) –
- **algo** (*str*) –
- **env_id** (*str*) –
- **log_folder** (*str*) –
- **tensorboard_log** (*str*) –
- **n_timesteps** (*int*) –
- **eval_freq** (*int*) –
- **n_eval_episodes** (*int*) –
- **save_freq** (*int*) –
- **hyperparams** (*Dict[str, Any] | None*) –
- **env_kwargs** (*Dict[str, Any] | None*) –
- **eval_env_kwargs** (*Dict[str, Any] | None*) –
- **trained_agent** (*str*) –

- **optimize_hyperparameters** (*bool*) –
- **storage** (*str | None*) –
- **study_name** (*str | None*) –
- **n_trials** (*int*) –
- **max_total_trials** (*int | None*) –
- **n_jobs** (*int*) –
- **sampler** (*str*) –
- **pruner** (*str*) –
- **optimization_log_path** (*str | None*) –
- **n_startup_trials** (*int*) –
- **n_evaluations** (*int*) –
- **truncate_last_trajectory** (*bool*) –
- **uuid_str** (*str*) –
- **seed** (*int*) –
- **log_interval** (*int*) –
- **save_replay_buffer** (*bool*) –
- **verbose** (*int*) –
- **vec_env_type** (*str*) –
- **n_eval_envs** (*int*) –
- **no_optim_plots** (*bool*) –
- **device** (*device | str*) –
- **config** (*str | None*) –
- **show_progress** (*bool*) –

create_envs(*n_envs*, *eval_env=False*, *no_log=False*)

Create the environment and wrap it if necessary.

Parameters

- **n_envs** (*int*) –
- **eval_env** (*bool*) – Whether is it an environment used for evaluation or not
- **no_log** (*bool*) – Do not log training when doing hyperparameter optim (issue with writing the same file)

Returns

the vectorized environment, with appropriate wrappers

Return type

VecEnv

learn(*model*)

Parameters

model (*BaseAlgorithm*) – an initialized RL model

Return type
None

`save_trained_model`(*model*)

Save trained model optionally with its replay buffer and `VecNormalize` statistics

Parameters
model (*BaseAlgorithm*) –

Return type
None

`setup_experiment`()

Read hyperparameters, pre-process them (create schedules, wrappers, callbacks, action noise objects) create the environment and possibly the model.

Returns
the initialized RL model

Return type
Tuple[*BaseAlgorithm*, *Dict*[str, *Any*]] | None

CHAPTER

TWELVE

WRAPPERS

class rl_zoo3.wrappers.**ActionNoiseWrapper**(*env*, *noise_std=0.1*)

Add gaussian noise to the action (without telling the agent), to test the robustness of the control.

Parameters

- **env** –
- **noise_std** – Standard deviation of the noise

step(*action*)

Uses the *step()* of the env that can be overwritten to change the returned data.

Parameters
action (*ndarray*) –

Return type
Tuple[*ObsType*, *SupportsFloat*, bool, bool, *Dict*[str, *Any*]]

class rl_zoo3.wrappers.**ActionSmoothingWrapper**(*env*, *smoothing_coef=0.0*)

Smooth the action using exponential moving average.

Parameters

- **env** –
- **smoothing_coef** – Smoothing coefficient (0 no smoothing, 1 very smooth)

reset(*seed=None*, *options=None*)

Uses the *reset()* of the env that can be overwritten to change the returned data.

Parameters

- **seed** (*int | None*) –
- **options** (*dict | None*) –

Return type
Tuple[*Tuple* | *Dict*[str, *Any*] | *ndarray* | int, *Dict*]

step(*action*)

Uses the *step()* of the env that can be overwritten to change the returned data.

Return type
Tuple[*Tuple* | *Dict*[str, *Any*] | *ndarray* | int, float, bool, bool, *Dict*]

class rl_zoo3.wrappers.**DelayedRewardWrapper**(*env*, *delay=10*)

Delay the reward by *delay* steps, it makes the task harder but more realistic. The reward is accumulated during those steps.

Parameters

- **env** –
- **delay** – Number of steps the reward should be delayed.

reset(*seed=None*, *options=None*)

Uses the *reset()* of the env that can be overwritten to change the returned data.

Parameters

- **seed** (*int | None*) –
- **options** (*dict | None*) –

Return type
Tuple[*Tuple* | *Dict*[str, *Any*] | *ndarray* | int, *Dict*]

step(*action*)

Uses the *step()* of the env that can be overwritten to change the returned data.

Return type
Tuple[*Tuple* | *Dict*[str, *Any*] | *ndarray* | int, float, bool, bool, *Dict*]

class rl_zoo3.wrappers.**FrameSkip**(*env*, *skip=4*)

Return only every skip-th frame (frameskipping)

Parameters

- **env** – the environment
- **skip** – number of skip-th frame

step(*action*)

Step the environment with the given action Repeat action, sum reward.

Parameters
action – the action

Returns
observation, reward, terminated, truncated, information

Return type
Tuple[*Tuple* | *Dict*[str, *Any*] | *ndarray* | int, float, bool, bool, *Dict*]

class rl_zoo3.wrappers.**HistoryWrapper**(*env*, *horizon=2*)

Stack past observations and actions to give an history to the agent.

Parameters

- **env** –
- **horizon** – Number of steps to keep in the history.

reset(*seed=None*, *options=None*)

Uses the *reset()* of the env that can be overwritten to change the returned data.

Parameters

- **seed** (*int | None*) –
- **options** (*dict | None*) –

Return type
Tuple[*ndarray*, *Dict*]

step(*action*)

Uses the *step()* of the env that can be overwritten to change the returned data.

Return type
Tuple[*ndarray*, *SupportsFloat*, bool, bool, *Dict*]

class rl_zoo3.wrappers.**HistoryWrapperObsDict**(*env*, *horizon=2*)

History Wrapper for dict observation.

Parameters

- **env** –
- **horizon** – Number of steps to keep in the history.

reset(*seed=None*, *options=None*)

Uses the *reset()* of the env that can be overwritten to change the returned data.

Parameters

- **seed** (*int | None*) –
- **options** (*dict | None*) –

Return type
Tuple[*Dict*[str, *ndarray*], *Dict*]

step(*action*)

Uses the *step()* of the env that can be overwritten to change the returned data.

Return type
Tuple[*Dict*[str, *ndarray*], *SupportsFloat*, bool, bool, *Dict*]

class rl_zoo3.wrappers.**MaskVelocityWrapper**(*env*)

Gym environment observation wrapper used to mask velocity terms in observations. The intention is the make the MDP partially observable. Adapted from https://github.com/LiuWenlin595/FinalProject.

Parameters
env – Gym environment

observation(*observation*)

Returns a modified observation.

Args:
observation: The env observation

Returns:
The modified observation

Parameters
observation (*ndarray*) –

Return type
ndarray

class rl_zoo3.wrappers.**TruncatedOnSuccessWrapper**(*env*, *reward_offset=0.0*, *n_successes=1*)

Reset on success and offsets the reward. Useful for GoalEnv.

reset(*seed=None*, *options=None*)

Uses the *reset()* of the env that can be overwritten to change the returned data.

Parameters

- **seed** (*int | None*) –
- **options** (*dict | None*) –

Return type
Tuple[*Tuple* | *Dict*[str, *Any*] | *ndarray* | int, *Dict*]

step(*action*)

Uses the *step()* of the env that can be overwritten to change the returned data.

Return type
Tuple[*Tuple* | *Dict*[str, *Any*] | *ndarray* | int, float, bool, bool, *Dict*]

CHAPTER

THIRTEEN

CALLBACKS

class rl_zoo3.callbacks.**ParallelTrainCallback**(*gradient_steps=100*, *verbose=0*, *sleep_time=0.0*)

Callback to explore (collect experience) and train (do gradient steps) at the same time using two separate threads. Normally used with off-policy algorithms and *train_freq=(1, "episode")*.

TODO: - blocking mode: wait for the model to finish updating the policy before collecting new experience at the end of a rollout - force sync mode: stop training to update to the latest policy for collecting new experience

Parameters

- **gradient_steps** (*int*) – Number of gradient steps to do before sending the new policy
- **verbose** (*int*) – Verbosity level
- **sleep_time** (*float*) – Limit the fps in the thread collecting experience.

class rl_zoo3.callbacks.**RawStatisticsCallback**(*verbose=0*)

Callback used for logging raw episode data (return and episode length).

class rl_zoo3.callbacks.**SaveVecNormalizeCallback**(*save_freq*, *save_path*, *name_prefix=None*, *verbose=0*)

Callback for saving a VecNormalize wrapper every `save_freq` steps

Parameters

- **save_freq** (*int*) – (int)
- **save_path** (*str*) – (str) Path to the folder where `VecNormalize` will be saved, as `vecnormalize.pkl`
- **name_prefix** (*str | None*) – (str) Common prefix to the saved `VecNormalize`, if None (default) only one file will be kept.
- **verbose** (*int*) –

class rl_zoo3.callbacks.**TrialEvalCallback**(*eval_env*, *trial*, *n_eval_episodes=5*, *eval_freq=10000*, *deterministic=True*, *verbose=0*, *best_model_save_path=None*, *log_path=None*)

Callback used for evaluating and reporting a trial.

Parameters

- **eval_env** (*VecEnv*) –
- **trial** (*Trial*) –
- **n_eval_episodes** (*int*) –
- **eval_freq** (*int*) –

- **deterministic** (*bool*) –
- **verbose** (*int*) –
- **best_model_save_path** (*str | None*) –
- **log_path** (*str | None*) –

CHAPTER

FOURTEEN

UTILS

class rl_zoo3.utils.**StoreDict**(*option_strings*, *dest*, *nargs=None*, ***kwargs*)

Custom argparse action for storing dict.

In: args1:0.0 args2:"dict(a=1)" Out: {'args1': 0.0, arg2: dict(a=1)}

rl_zoo3.utils.**create_test_env**(*env_id*, *n_envs=1*, *stats_path=None*, *seed=0*, *log_dir=None*, *should_render=True*, *hyperparams=None*, *env_kwargs=None*)

Create environment for testing a trained agent

Parameters

- **env_id** (*str*) –
- **n_envs** (*int*) – number of processes
- **stats_path** (*str | None*) – path to folder containing saved running averaged
- **seed** (*int*) – Seed for random number generator
- **log_dir** (*str | None*) – Where to log rewards
- **should_render** (*bool*) – For Pybullet env, display the GUI
- **hyperparams** (*Dict[str, Any] | None*) – Additional hyperparams (ex: n_stack)
- **env_kwargs** (*Dict[str, Any] | None*) – Optional keyword argument to pass to the env constructor

Returns

Return type
VecEnv

rl_zoo3.utils.**get_callback_list**(*hyperparams*)

Get one or more Callback class specified as a hyper-parameter "callback". e.g. callback: stable_baselines3.common.callbacks.CheckpointCallback

for multiple, specify a list:

callback:

- rl_zoo3.callbacks.PlotActionWrapper
- stable_baselines3.common.callbacks.CheckpointCallback

Parameters
hyperparams (*Dict[str, Any]*) –

Returns

Return type
List[*BaseCallback*]

rl_zoo3.utils.**get_class_by_name**(*name*)

Imports and returns a class given the name, e.g. passing 'stable_baselines3.common.callbacks.CheckpointCallback' returns the CheckpointCallback class.

Parameters
name (*str*) –

Returns

Return type
Type

rl_zoo3.utils.**get_hf_trained_models**(*organization='sb3'*, *check_filename=False*)

Get pretrained models, available on the Hugginface hub for a given organization.

Parameters

- **organization** (*str*) – Huggingface organization Stable-Baselines (SB3) one is the default.
- **check_filename** (*bool*) – Perform additional check per model to be sure they match the RL Zoo convention. (this will slow down things as it requires one API call per model)

Returns
Dict representing the trained agents

Return type
Dict[str, *Tuple*[str, str]]

rl_zoo3.utils.**get_latest_run_id**(*log_path*, *env_name*)

Returns the latest run number for the given log name and log path, by finding the greatest number in the directories.

Parameters

- **log_path** (*str*) – path to log folder
- **env_name** (*EnvironmentName*) –

Returns
latest run number

Return type
int

rl_zoo3.utils.**get_saved_hyperparams**(*stats_path*, *norm_reward=False*, *test_mode=False*)

Retrieve saved hyperparameters given a path. Return empty dict and None if the path is not valid.

Parameters

- **stats_path** (*str*) –
- **norm_reward** (*bool*) –
- **test_mode** (*bool*) –

Returns

Return type
Tuple[*Dict*[str, *Any*], str | None]

rl_zoo3.utils.**get_trained_models**(*log_folder*)

Parameters
log_folder (*str*) – Root log folder

Returns
Dict representing the trained agents

Return type
Dict[str, *Tuple*[str, str]]

rl_zoo3.utils.**get_wrapper_class**(*hyperparams*, *key='env_wrapper'*)

Get one or more Gym environment wrapper class specified as a hyper parameter "env_wrapper". Works also for VecEnvWrapper with the key "vec_env_wrapper".

e.g. env_wrapper: gym_minigrid.wrappers.FlatObsWrapper

for multiple, specify a list:

env_wrapper:

- rl_zoo3.wrappers.PlotActionWrapper
- rl_zoo3.wrappers.TimeFeatureWrapper

Parameters

- **hyperparams** (*Dict[str, Any]*) –
- **key** (*str*) –

Returns
maybe a callable to wrap the environment with one or multiple gym.Wrapper

Return type
Callable[[*Env*], *Env*] | None

rl_zoo3.utils.**linear_schedule**(*initial_value*)

Linear learning rate schedule.

Parameters
initial_value (*float | str*) – (float or str)

Returns
(function)

Return type
Callable[[float], float]

CHAPTER

FIFTEEN

CHANGELOG

See https://github.com/DLR-RM/rl-baselines3-zoo/blob/master/CHANGELOG.md

CHAPTER

SIXTEEN

CITING RL BASELINES3 ZOO

To cite this project in publications:

```
@misc{rl-zoo3,
  author = {Raffin, Antonin},
  title = {RL Baselines3 Zoo},
  year = {2020},
  publisher = {GitHub},
  journal = {GitHub repository},
  howpublished = {\url{https://github.com/DLR-RM/rl-baselines3-zoo}},
}
```

CHAPTER

SEVENTEEN

CONTRIBUTING

To any interested in making the rl baselines better, there are still some improvements that need to be done. You can check issues in the repo.

If you want to contribute, please read CONTRIBUTING.md first.

CHAPTER

EIGHTEEN

INDICES AND TABLES

- genindex
- search
- modindex

PYTHON MODULE INDEX

r

INDEX

T