MALib

Release v0.1.0

SJTU-MARL

Jul 26, 2023

TUTORIALS

1 Introduction **1**
- 1.1 Feature Overview . . . 1
- 1.2 Citing MALib . . . 1

2 Quick Start **3**
- 2.1 Installation . . . 3
- 2.2 An Example: Policy Space Response Oracles . . . 4
- 2.3 Support Traditional (MA)RL . . . 7

3 Key Concepts **9**
- 3.1 Scenarios . . . 9
- 3.2 Reinforcement Learning Algorithms . . . 10
- 3.3 Rollout Management . . . 12
- 3.4 Population Evaluation . . . 13

4 Distributed Strategies **15**
- 4.1 Bulk Synchronous Parallel(BSP) . . . 15
- 4.2 Bounded Asynchronous Parallel(BAP) . . . 17

5 Environments **19**
- 5.1 Available Environments . . . 19
- 5.2 Environment Customiztion . . . 22

6 Clustered Deployment **25**
- 6.1 Start the Head Node . . . 25
- 6.2 Start Worker Nodes . . . 25
- 6.3 Dashboard . . . 27

7 malib.agent package **29**
- 7.1 Submodules . . . 29
- 7.2 malib.agent.agent_interface module . . . 29
- 7.3 malib.agent.async_agent module . . . 32
- 7.4 malib.agent.indepdent_agent module . . . 32
- 7.5 malib.agent.manager module . . . 33
- 7.6 malib.agent.team_agent module . . . 35

8 malib.backend package **37**
- 8.1 Submodules . . . 37
- 8.2 malib.backend.offline_dataset_server module . . . 37
- 8.3 malib.backend.parameter_server module . . . 38

9 malib.common package **41**
9.1 Submodules . . . 41
9.2 malib.common.distributions module . . . 41
9.3 malib.common.manager module . . . 51
9.4 malib.common.payoff_manager module . . . 52
9.5 malib.common.strategy_spec module . . . 52

10 malib.models package **55**
10.1 Subpackages . . . 55

11 malib.remote package **67**
11.1 Submodules . . . 67
11.2 malib.remote.interface module . . . 67

12 malib.rl package **69**
12.1 Subpackages . . . 69

13 malib.rollout package **83**
13.1 Subpackages . . . 83
13.2 Submodules . . . 92
13.3 malib.rollout.manager module . . . 92
13.4 malib.rollout.pb_rolloutworker module . . . 92
13.5 malib.rollout.rolloutworker module . . . 92

14 malib.scenarios package **93**
14.1 Submodules . . . 93
14.2 malib.scenarios.league_training_scenario module . . . 93
14.3 malib.scenarios.marl_scenario module . . . 93
14.4 malib.scenarios.psro_scenario module . . . 93
14.5 malib.scenarios.scenario module . . . 93

15 malib.utils package **95**
15.1 Submodules . . . 95
15.2 malib.utils.data module . . . 95
15.3 malib.utils.episode module . . . 95
15.4 malib.utils.exploitability module . . . 97
15.5 malib.utils.general module . . . 98
15.6 malib.utils.logging module . . . 101
15.7 malib.utils.monitor module . . . 101
15.8 malib.utils.notations module . . . 102
15.9 malib.utils.preprocessor module . . . 102
15.10 malib.utils.replay_buffer module . . . 104
15.11 malib.utils.schedules module . . . 104
15.12 malib.utils.statistic module . . . 105
15.13 malib.utils.stopping_conditions module . . . 106
15.14 malib.utils.tasks_register module . . . 106
15.15 malib.utils.tianshou_batch module . . . 106
15.16 malib.utils.timing module . . . 109
15.17 malib.utils.typing module . . . 109

16 Contributing to MALib **111**
16.1 Code of Conduct . . . 111
16.2 Setup Development Environment . . . 111
16.3 Where to Get Started . . . 112
16.4 Committing . . . 112

16.5 Pre-Push Checklist . . . 112
16.6 Submission of a Pull Request . . . 112
16.7 Communication . . . 113
16.8 Feature Requests . . . 113
16.9 Formatting . . . 113

17 Changelog **115**

18 License **117**

Python Module Index **119**

Index **121**

CHAPTER

ONE

INTRODUCTION

MALib is a parallel framework for population-based learning methods including Policy Space Response Oracle, Self-Play, Neural Fictitious Self-Play, etc. which are nested with (multi-agent) reinforcement learning algorithms. MALib provides higher-level abstractions of MARL training paradigms, enabling efficient code reuse and flexible deployments on distributed strategies. The design of MALib also strives to promote the research of other multi-agent learning research, including multi-agent imitation learning and model-based RL.

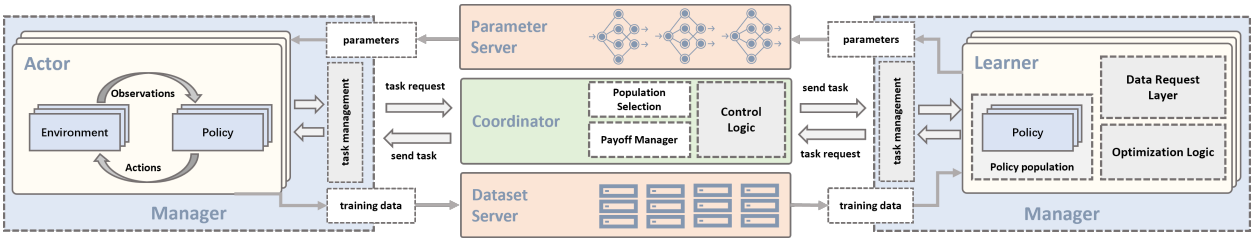

Fig. 1: Overview of the MALib architecture.

1.1 Feature Overview

The key features of MALib are listed as follows:

- *Key Concepts*
- marl-abstraction-doc
- *Distributed Strategies*
- *Environments*

1.2 Citing MALib

If you use MALib in your work, please cite the accompanying paper.

```
@inproceedings{zhou2021malib,
    title={MALib: A Parallel Framework for Population-based Multi-agent Reinforcement␣
↪Learning},
    author={Zhou, Ming and Wan, Ziyu and Wang, Hanjing and Wen, Muning and Wu, Runzhe␣
↪and Wen, Ying and Yang, Yaodong and Zhang, Weinan and Wang, Jun},
    booktitle={Preprint},
```

(continues on next page)

(continued from previous page)

```
    year={2021},
    organization={Preprint}
}
```

CHAPTER

TWO

QUICK START

Follow this page to install your MALib and try an example training case.

2.1 Installation

MALib has been tested on Python 3.7 and above. The system requirement is ubuntu18.04 or above. Windows is still not supported.

2.1.1 Conda Environment

We strongly recommend using conda to manage your dependencies, and avoid version conflicts. Here we show the example of building python 3.7 based conda environment.

```
conda create -n malib python==3.7 -y
conda activate malib

# install dependencies
cmake --version # must be >=3.12
clang++ --version   # must be >=7.0.0
sudo apt-get install graphviz cmake clang

# install malib
pip install -e .
```

2.1.2 Setup Development Environment

For users who wanna contribute to our repository, run `pip install -e .[dev]` to complete the development dependencies, also refer to the section *Contributing to MALib*.

2.2 An Example: Policy Space Response Oracles

A typical population-based algorithm that MALib supports is Policy Space Response Oracles (PSRO). In this section, we give an example of PSRO to show how to start a population-based training case.

2.2.1 Overview

PSRO cooperates empirical game-theoretical analysis and nested (multi-agent) reinforcement learning algorithms to solve multi-agent learning tasks in the scope of meta-game. At each iteration, the algorithm will generate some policy combinations and executes reinforcement learning to compute best responses for each agent. Such a nested learning process comprises rollout, training, evaluation in sequence, and works circularly until the algorithm finds the estimated Nash Equilibrium. The following picture gives an overview of the learning process of PSRO.

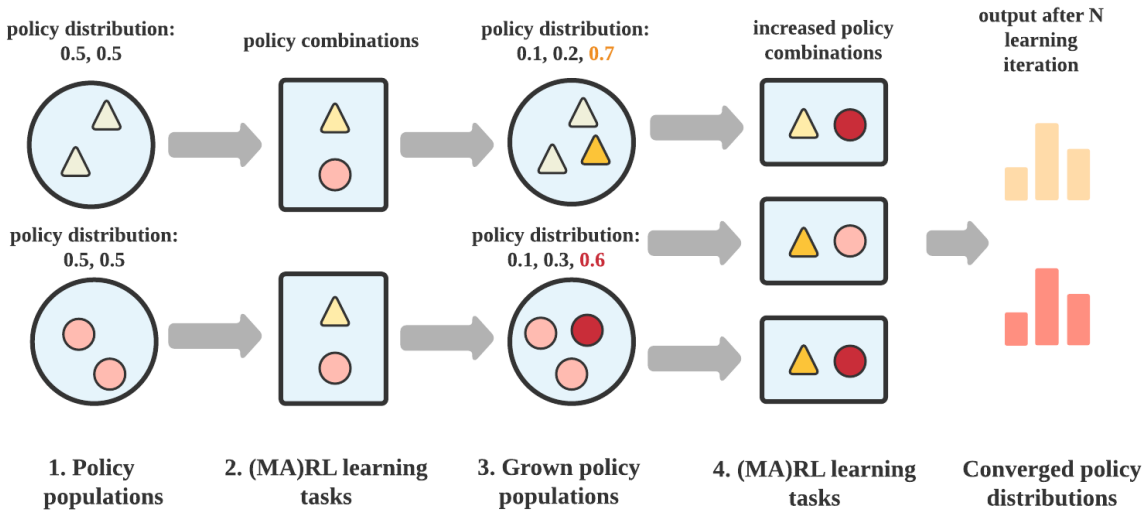

Fig. 1: Overview of the learning process of PSRO

Though the training workflow seems so complex in speaking (and the above illustration), MALib divides it into several independent components. Therefore, you can quickly launch such training with lines of code as follows.

2.2.2 Setup Underlying RL Algorithm

PSRO requires an underlying RL algorithm to find the best response at each learning iteration, so the first step is to determine which RL algorithm you want to use in your training.

```
from malib.rl.dqn import DQNPolicy, DQNTrainer

algorithms = {
    "default": (
        DQNPolicy,
        DQNTrainer,
        # model configuration, None for default
        {},
```

(continues on next page)

(continued from previous page)

```
        {},
    )
}
```

MALib integrates many kinds of (MA)RL algorithms, which follow the policy and model interfaces as Tianshou. Thus, users can easily migrate and test standard RL algorithms in population-based learning cases, free from RL algorithm reproduction. In this example, we choose Deep Q-learning Networks (DQN).

A key concept in MALib is that the devide of training paradigms, policy behavior and loss computation. The above algorithm configuration includes both `DQNPolicy` and `DQNTrainer`, they are implemented for policy behavior definition and loss computation respectively. As the policy and loss coniguration has been given, the next thing is to determine the training paradigm. Since the DQN is an independent learning algorithm, we use `IndependentAgent` as the best choice as follow:

```
from malib.rl.dqn import DEFAULT_CONFIG
from malib.agent import IndependentAgent

trainer_config = DEFAULT_CONFIG["training_config"].copy()
trainer_config["total_timesteps"] = int(1e6)

training_config = {
    "type": IndependentAgent,
    "trainer_config": trainer_config,
    "custom_config": {},
}
```

Users can also implement their own algorithms and cooperate with the existing training paradigms in MALib. To understand how to do that, you can refer to marl-abstraction-doc.

2.2.3 Setup Environment

The the environment is setup as follow:

```
from malib.rollout.envs.open_spiel import env_desc_gen

env_description = env_desc_gen(env_id="kuhn_poker")
```

2.2.4 Setup the Rollout

After you've determined the underlying RL algorithm and the environment, another key step is to determine the rollout configuration. In MALib, the rollout procedure is fully independent to the policy optimization, and performs asynchronous. To configure the rollout procedure for PSRO training, the users can create a coniguration as below:

```
rollout_config = {
    "fragment_length": 2000,  # every thread
    "max_step": 200,
    "num_eval_episodes": 10,
    "num_threads": 2,
    "num_env_per_thread": 10,
    "num_eval_threads": 1,
    "use_subproc_env": False,
```

(continues on next page)

(continued from previous page)

```
    "batch_mode": "time_step",
    "postprocessor_types": ["defaults"],
    # every # rollout epoch run evaluation.
    "eval_interval": 1,
    "inference_server": "ray",
}
```

Most of the keys in `rollout_config` are used to determine the rollout parallelism, e.g., `num_env_per_thread`, `num_eval_threads` and `use_subproc_env`. As for the `inference_server`, it determines what kind of inference mechanism will be used. Currently, we only open the use of Ray-based. For more details about the configuration of rollout, please refer to rollout-doc.

2.2.5 Train PSRO with a Scenario

Pack all of the above setup as a scenario, then start the learning by loading it to `run`:

```
import time

from malib.runner import run
from malib.scenarios.psro_scenario import PSROScenario

env_description = env_desc_gen(env_id="kuhn_poker")
runtime_logdir = os.path.join("./logs", f"psro_kuhn_poker/{time.time()}")

if not os.path.exists(runtime_logdir):
    os.makedirs(runtime_logdir)

scenario = PSROScenario(
    name="psro_kuhn_poker",
    log_dir=runtime_logdir,
    algorithms=algorithms,
    env_description=env_description,
    training_config=training_config,
    rollout_config=rollout_config,
    # control the outer loop.
    global_stopping_conditions={"max_iteration": 50},
    agent_mapping_func=agent_mapping_func,
    # for the training of best response.
    stopping_conditions={
        "training": {"max_iteration": int(1e4)},
        "rollout": {"max_iteration": 100},
    },
)

run(scenario)
```

2.3 Support Traditional (MA)RL

Similar to the above example. Users can run traditional (multi-agent) reinforcement learning algorithms with MALib:

```
import os
import time

from malib.runner import run
from malib.agent import IndependentAgent
from malib.scenarios.marl_scenario import MARLScenario
from malib.rl.dqn import DQNPolicy, DQNTrainer, DEFAULT_CONFIG
from malib.rollout.envs.gym import env_desc_gen

trainer_config = DEFAULT_CONFIG["training_config"].copy()
trainer_config["total_timesteps"] = int(1e6)

training_config = {
    "type": IndependentAgent,
    "trainer_config": trainer_config,
    "custom_config": {},
}

rollout_config = {
    "fragment_length": 2000,  # determine the size of sended data block
    "max_step": 200,
    "num_eval_episodes": 10,
    "num_threads": 2,
    "num_env_per_thread": 10,
    "num_eval_threads": 1,
    "use_subproc_env": False,
    "batch_mode": "time_step",
    "postprocessor_types": ["defaults"],
    # every # rollout epoch run evaluation.
    "eval_interval": 1,
    "inference_server": "ray",  # three kinds of inference server: `local`, `pipe` and␣
↪`ray`
}
agent_mapping_func = lambda agent: agent

algorithms = {
    "default": (
        DQNPolicy,
        DQNTrainer,
        # model configuration, None for default
        {},
        {},
    )
}

env_description = env_desc_gen(env_id="CartPole-v1", scenario_configs={})
runtime_logdir = os.path.join("./logs", f"gym/{time.time()}")

if not os.path.exists(runtime_logdir):
```

(continues on next page)

(continued from previous page)

```
    os.makedirs(runtime_logdir)

scenario = MARLScenario(
    name="gym",
    log_dir=runtime_logdir,
    algorithms=algorithms,
    env_description=env_description,
    training_config=training_config,
    rollout_config=rollout_config,
    agent_mapping_func=agent_mapping_func,
    stopping_conditions={
        "training": {"max_iteration": int(1e10)},
        "rollout": {"max_iteration": 1000, "minimum_reward_improvement": 1.0},
    },
)

run(scenario)
```

CHAPTER

THREE

KEY CONCEPTS

This page will help you to understand the workflow of MALib to train a population-based reinforcement learning algorithm. As for implementing such an algorithm instance, key components including `Policy`, `Evaluator`, `RolloutWorkerManager` and `AgentInterfaceManager`. Functionally, the `AgentInterfaceManager` is responsible for a cluster of `AgentInterface`, while the `RolloutWorkerManager` for a cluster of `RolloutWorker`. The `Policy` is implemented as a behavior interface that packs models that parameterize an agent policy. As for the nested reinforcement learning algorithm, we depart it as a coordination of `AgentInterfaceManager` and `RolloutWorkerManager`. We pack all of them as a scenario which isolates the details of components interaction.

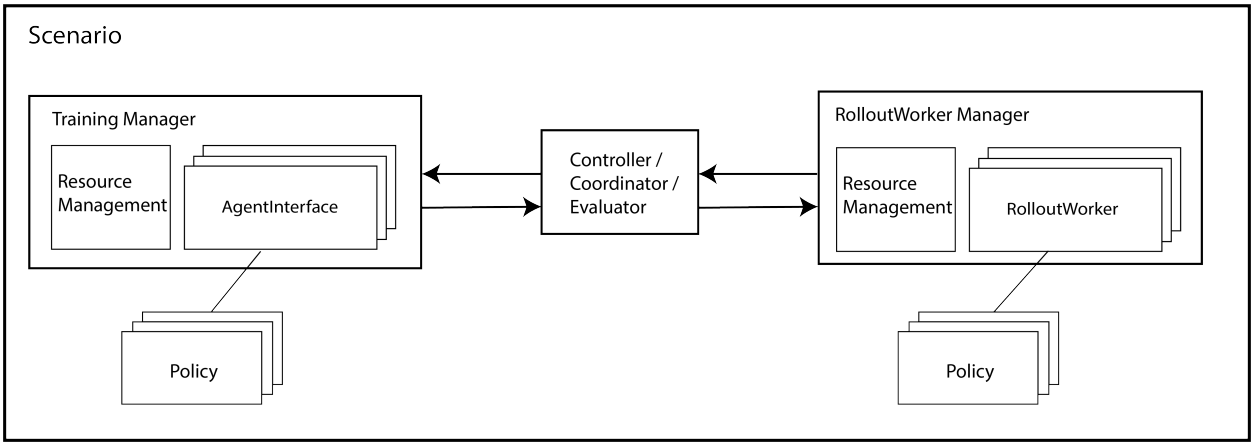

Fig. 1: Overview of the interaction between Managers

3.1 Scenarios

A scenario defines indicates a training instance, it brings all MALib components and the resource management together. Users can create their own scenario by inheriting the `Scenario` class, as we have implemented two standard scenarios under the *malib.scenarios package*. To deploy a scenario as an instance, you only need to implement an scenario instance like `PSROScenario`, then load it to the runner interface that locates under *malib.runner*. An example is listed as follow

```
from malib.runner import run
from malib.scenarios.psro_scenario import PSROScenario

scenario = PSROScenario(
    name=f"psro_{env_id}",
    log_dir=runtime_logdir,
```

(continues on next page)

(continued from previous page)

```
    algorithms=algorithms,
    env_description=env_description,
    training_config=training_config,
    rollout_config=rollout_config,
    # control the outer loop.
    global_stopping_conditions={"max_iteration": 50},
    agent_mapping_func=agent_mapping_func,
    # for the training of best response.
    stopping_conditions={
        "training": {"max_iteration": int(1e4)},
        "rollout": {"max_iteration": 100},
    },
)

run(scenario)
```

3.1.1 Available Scenarios

- marl-scenario-doc
- psro-scenario-doc
- league-training-doc

3.2 Reinforcement Learning Algorithms

MALib supports population-based learning algorithms that run nested reinforcement learning process. To better coordinate with the high-level population-based optimization, MALib devides traditional reinforcement learning algorithms into three key concepts, i.e., Policy, Trainer and AgentInterface.

3.2.1 Policy

In a nutshell, policies are Python classes that define how an agent acts in an environment. Agents query the policy to determine actions. In an environment, there would be a multiple policies and some of them can be linked to multiple environment agents.

Currently, the implementation of policies is compatible with Tianshou library. However, for PyTorch implementation only. The customization of policies is very convenient for users, as we've abstract the policies into two mainstream implementation, i.e., value-based and policy-gradient-based. For example, the implementation of A2C could be:

```
class A2CPolicy(PGPolicy):
    def __init__(
        self,
        observation_space: spaces.Space,
        action_space: spaces.Space,
        model_config: Dict[str, Any],
        custom_config: Dict[str, Any],
        **kwargs
    ):
        super().__init__(
```

(continues on next page)

(continued from previous page)

```
            observation_space, action_space, model_config, custom_config, **kwargs
        )

        preprocess_net: nn.Module = self.actor.preprocess
        if isinstance(action_space, spaces.Discrete):
            self.critic = discrete.Critic(
                preprocess_net=preprocess_net,
                hidden_sizes=model_config["hidden_sizes"],
                device=self.device,
            )
        elif isinstance(action_space, spaces.Box):
            self.critic = continuous.Critic(
                preprocess_net=preprocess_net,
                hidden_sizes=model_config["hidden_sizes"],
                device=self.device,
            )
        else:
            raise TypeError(
                "Unexpected action space type: {}".format(type(action_space))
            )

        self.register_state(self.critic, "critic")

    def value_function(self, observation: torch.Tensor, evaluate: bool, **kwargs):
        """Compute values of critic."""

        with torch.no_grad():
            values, _ = self.critic(observation)
        return values.cpu().numpy()
```

3.2.2 Trainer

A `Trainer` defines the loss computation and specific training logics for a policy, users can load a policy instance and training configuration to peform training.

```
from mailb.rl.dqn import DQNTrainer, DEFAULT_CONFIG

trainer = DQNTrainer(
    training_config=DEFAULT_CONFIG["training_config"],
    policy_instance=policy
)

loss = trainer(buffer=Batch(**data))
```

See *malib.rl.common package* to get more details about the customization of trainer.

3.2.3 AgentInterface

Conceptually, an `AgentInterface` manages a policy pool and its dependencies. Most importantly, schedule policy training according to the current policy combination. `AgentInterface` does not execute the specific training logic but pulls training data from the remote dataset server and syncs up policy parameters with the remote parameter server. It can also implement different training paradigms and distributed strategies. See *Distributed Strategies* to get more details.

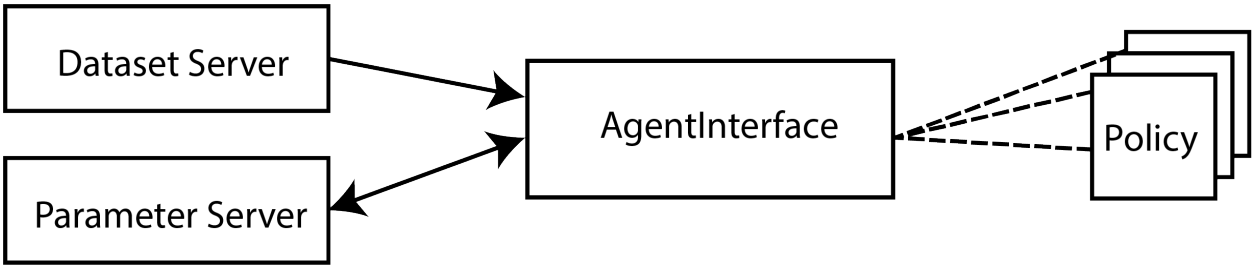

Fig. 2: Overview of the interaction between Managers

3.2.4 AgentInterface Management

In the case of population-based and multi-agent learning, the basic management unit would be a policy pool, and there would be a training interface that is responsible for the training or evolution of each of them. As we observed, in most existing population-based RL algorithms, the training of each agent is often isolated, i.e., no interaction between the populations in the training stage. The management of training policies is implemented as `TrainingManager` in *malib.agent package*. In multi-agent cases, there would be multiple simultaneous training job for the agents. As we've introduced the mechanism of `RolloutWorkerManger` in previous section, each `AgentInterface` has at least one `RolloutWorker`.

3.3 Rollout Management

The management of rollout workers is implemented as `RolloutWorkerManger` in *malib.rollout package*. As the training cases involve multiple agents, MALib creates independent rollout workers for each training agent to achieve as much efficiency as possible. Each `RolloutWorker` encapsulates an actor pool that contains multiople inference CS instance(s).

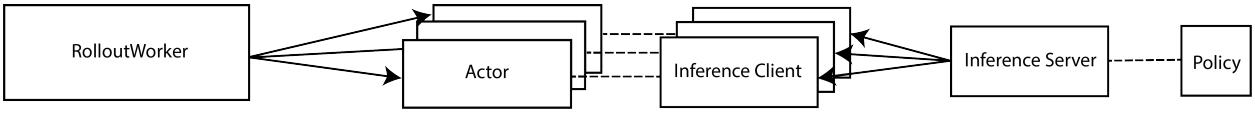

Fig. 3: Overview of the interaction between Managers

3.3.1 Rollout Worker

A rollout worker is responsible for the specific simulation tasks that disttributed from the higher controller. As the simulation tasks could be heterogenous on the policy combination and behavior (e.g., exploration mode for training data collection, and exploit mode for policy evaluation), an `RolloutWorker` creates an actor pool that considers both requirements of evaluation and data collection.

```
actor_pool = ActorPool(
    [
        self.inference_client_cls.remote(
            env_desc,
            ray.get_actor(settings.OFFLINE_DATASET_ACTOR),
            max_env_num=num_env_per_thread,
            use_subproc_env=rollout_config["use_subproc_env"],
            batch_mode=rollout_config["batch_mode"],
            postprocessor_types=rollout_config["postprocessor_types"],
            training_agent_mapping=agent_mapping_func,
        )
        for _ in range(num_threads + num_eval_threads)
    ]
)
```

Furthermore, as the number of episodes for evaluation or data collection could be large, then a single-thread environment simulation would cause many waiting fragments that harm the simulation performance overall. The **environment vectorization** technique is considered in the implementation of `RolloutWorker`, more details can be found in the *Environments* section. There are two kinds of policy use strategies for the interaction between polcies and environments, i.e., shared policy servers or independent copies of policies. MALib considers both of them in the implementation of `RolloutWorker`. See rollout-doc for more details.

3.4 Population Evaluation

The population evaluation is performed after some rounds of training, it is built upon the policy combination evlauation and meta-solvers-doc (for computing policy distribution over a population). To evaluation a given population, there would be many of policy combinations given by a payoff-manager-doc.

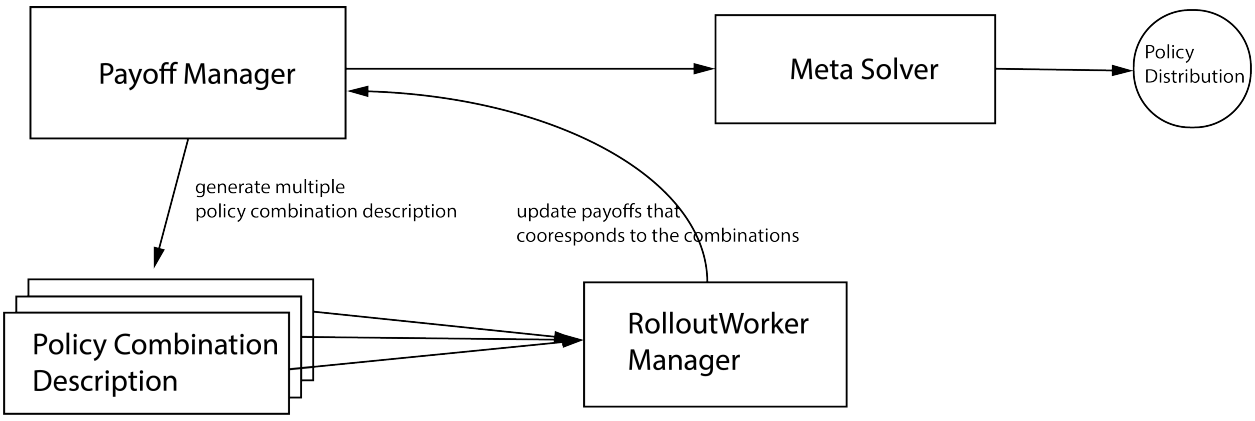

Fig. 4: Overview of population evaluation

CHAPTER

FOUR

DISTRIBUTED STRATEGIES

MALib has included typical distributed strategies that have been applied in existing distributed reinforcement learning algorithms. As MALib departs the implementation of rollout and training, it can easily distribute different distributed strategies by calling different `AgentInterface`.

AgentInterface is an esstential layer of abstraction in MALib's distributed architecture. Large-scale distributed training often requires expertise in tuning performance and convergence and thus users have to have strong backgrounds of both algorithms and distributed computing. As a trade-off between efficiency and flexibility, the high-level abstractions proposed by existing distributed RL frameworks either has limited scalability or fail to support more complicated training paradigm(e.g. population-based methods). With special focus on these issues, MALib introduces AgentInterface for fine-grained logic controlling and data communications. Logically, AgentInterface offers an unified interface of policy-data interaction for conventional RL and population-based training, while it can also be configured to be a local sink node managing parameter versions of sub-workers and offloading computation from the central node, offering efficient and highly-customizable interface for population-based training and large-scale training scenarios. MALib provides off-the-shelf implementations of AgentInterface convering several common distributed training paradigms.

4.1 Bulk Synchronous Parallel(BSP)

Under the BSP mode, the sub-workers managed by an AgentInterface are initialized with the same copy of model parameters. In each iteration, sub-workers execute local gradients computation, followed by gradients submission to the Parameter Server(PS) and a global synchronous parameters update from PS. The PS will not pub the updated version of parameters until it has received and aggregated gradients from all of the sub-workers, which means during the whole training process all sub-workers of an AgentInterface has strictly the same version of model parameters. BSP naturally extends algorithms to the distributed scenarios but can suffer from synchronization overheads caused by the global synchronization barriers, especially when sub-workers have uneven computational speeds, which is also knowns as the struggler problem.

Asynchronous Parallel(ASP) As opposite to the BSP mode, AgentInterface under ASP mode remove the synchronization barriers. All sub-workers submit local gradients to the PS, where gradients are applied to a globally shared copy of parameters, and pull the latest version of global parameters from the PS. ASP has better utilization of faster workers and less overheads due to asynchronous communication and parameter aggregration. However, ASP fails to offer theoretical guarantees of convergence.

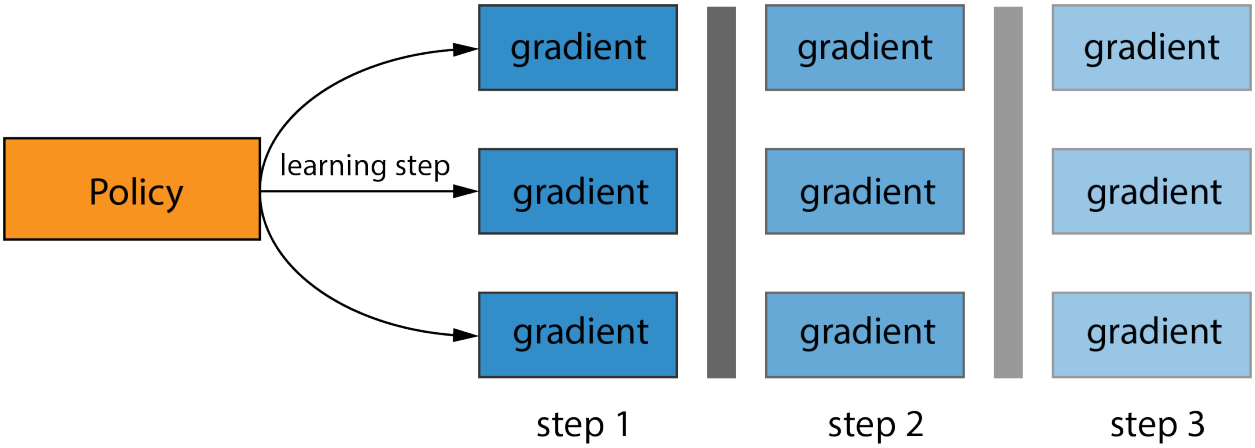

Fig. 1: Exmaple illustration of BSP

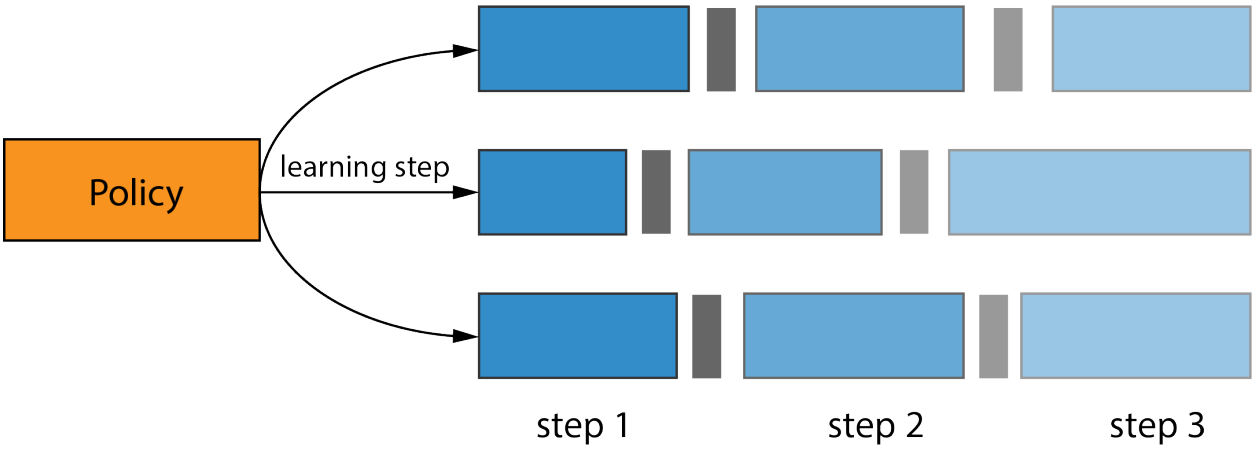

Fig. 2: Exmaple illustration of ASP

4.2 Bounded Asynchronous Parallel(BAP)

To alleviate the struggler problem in distributed computing while without being completely loss of synchronization, some machine learning algorithms adopts BAP model, which stands in the middle of the BSP and ASP models. The PS under BAP mode updates the global parameters only when it has received and aggregated all gradients from all sub-workers. Each sub-worker will check if a version of global parameters that is fresher than its local cache is available after submitting its local gradients generated from the previous iteration. The degree of staleness is defined as the difference in iteration numbers between the faster worker and the slowest worker. If fresher version of global parameters is ready for read, than sub-workers will pull the global parameters from the PS and update the local cache, while the stale copy of parameters from local cache(without updates from local gradients) is adopted if a newer version of global parameters is not available and the staleness of the sub-worker is less than the pre-defined staleness threshold S. Moreover, a threshold for longest living time of a single iteration is set, the violation of which will invoke an force interruption of ongoing iteration and a sync up, the result of iterrupted iteration(i.e. local gradients) will be ignored in the aggregation of certain iteration i.

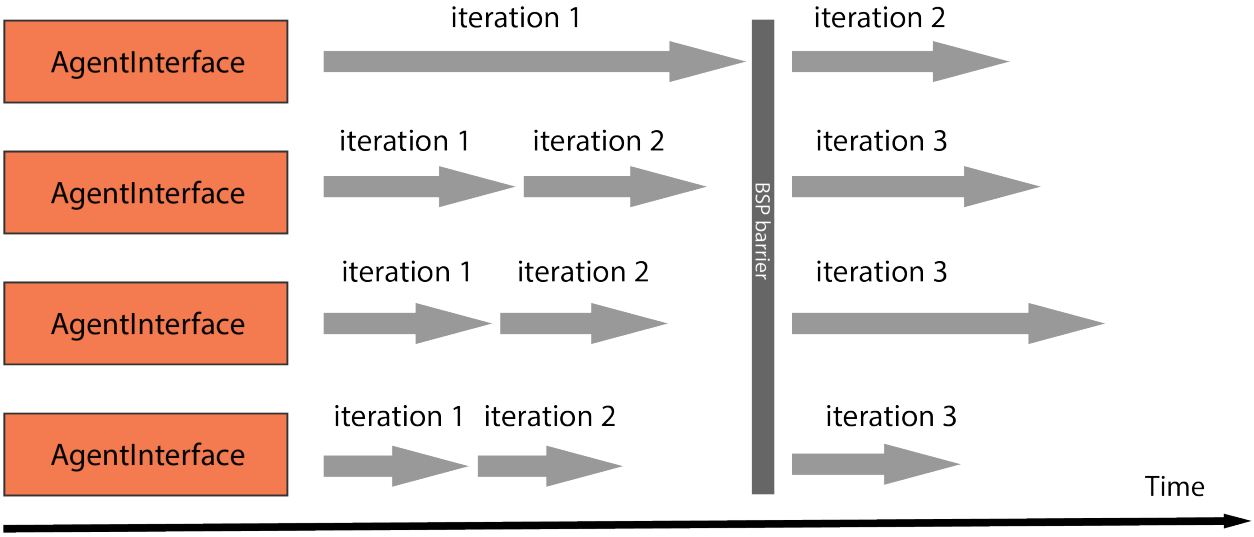

Fig. 3: Exmaple illustration of BAP.

CHAPTER

FIVE

ENVIRONMENTS

MALib implements a unified environment interface to satisfy both turn-based and simultaneous-based environments. MALib works with different environments, including simple Markov Decision Process environments, OpenAI-gym, OpenSpiel, and other user-defined environments under MALib's environment API. We first introduce the available environments supported by MALib and then give an example of how to customize your environments.

5.1 Available Environments

This section introduce the environments have been integrated in MALib.

5.1.1 Simple Markov Decision Process

mdp is a simple and easy-to-specify environment for standard Markov Decision Process. Users can create an instance as follows:

```
from malib.rollout.envs.mdp import MDPEnvironment, env_desc_gen

env = MDPEnvironment(env_id="one_round_mdp")

# or get environment description with `env_desc_gen`
env_desc = env_desc_gen(env_id="one_round_mdp")
# return an environment description as a dict:
# {
#     "creator": MDPEnvironment,
#     "possible_agents": env.possible_agents,
#     "action_spaces": env.action_spaces,
#     "observation_spaces": env.observation_spaces,
#     "config": {'env_id': env_id},
# }
```

Note: In MALib, this environment is used as a minimal testbed for verification of our algorithms' implementation. Users can use it for rapid algorithm validation.

The available scenarios including:

- `one_round_dmdp`: one-round deterministic MDP
- `two_round_dmdp`: two-round deterministic MDP
- `one_round_nmdp`: one-round stochastic MDP

- `two_round_nmdp`: two-round stochastic MDP
- `multi_round_nmdp`: multi-round stochastic MDP

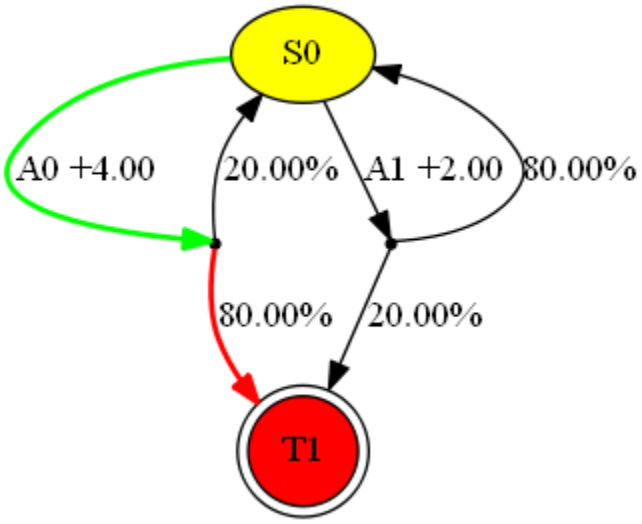

Fig. 1: Illustration of a Multi-round stochastic MDP

If you want to customize a MDP, you can follow the guides in the original repository.

5.1.2 OpenAI-Gym

Gym is an open-source Python library for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments and a standard set of environments compliant with that API. Since its release, Gym's API has become the field standard for doing this.

```
from malib.rollout.envs.gym import GymEnv, env_desc_gen

env = GymEnv(env_id="CartPole-v1", scenario_configs={})

env_desc = env_desc_gen(env_id="CartPole-v1", scenarios_configs={})
# return an environment description as a dict:
# {
#     "creator": GymEnv,
#     "possible_agents": env.possible_agents,
#     "action_spaces": env.action_spaces,
#     "observation_spaces": env.observation_spaces,
#     "config": config,
# }
```

5.1.3 DeepMind OpenSpiel

OpenSpiel is a collection of environments and algorithms for research in general reinforcement learning and search/planning in games. OpenSpiel supports n-player (single- and multi- agent) zero-sum, cooperative and general-sum, one-shot and sequential, strictly turn-taking and simultaneous-move, perfect and imperfect information games, as well as traditional multiagent environments such as (partially- and fully- observable) grid worlds and social dilemmas. OpenSpiel also includes tools to analyze learning dynamics and other common evaluation metrics. Games are represented as procedural extensive-form games, with some natural extensions.

```
from malib.rollout.envs.open_spiel import OpenSpielEnv, env_desc_gen

env = OpenSpielEnv(env_id="goofspiel")

env_desc = env_des_gen(env_id="goofspiel")
# return an environment description as a dict:
# {
#     "creator": OpenSpielEnv,
#     "possible_agents": env.possible_agents,
#     "action_spaces": env.action_spaces,
#     "observation_spaces": env.observation_spaces,
#     "config": config,
# }
```

5.1.4 PettingZoo

PettingZoo is a Python library for conducting research in multi-agent reinforcement learning, akin to a multi-agent *Gym environment <https://github.com/Farama-Foundation/Gymnasium>*. It integrates many popular multi-agent environments, also modified multi-agent Atari games.

Available Environments

- Atari: Multi-player Atari 2600 games (cooperative, competitive and mixed sum)
- Butterfly: Cooperative graphical games developed by us, requiring a high degree of coordination
- Classic: Classical games including card games, board games, etc.
- MPE: A set of simple nongraphical communication tasks, originally from https://github.com/openai/multiagent-particle-envs
- SISL: 3 cooperative environments, originally from https://github.com/sisl/MADRL

Note: For the use of multi-agent Atari in PettingZoo, you should run `AutoROM to install rom, and `pettiongzoo[classic]` to support Classic games; `pettingzoo[sisl]` to support SISL environments.

There is a file named `scenarios_configs_re.py` under the package of `malib.rollout.envs.pettingzoo` which offers a default dictionary of supported scenarios and configurations. Users can create a pettingzoo sub enviornment by giving an environment id in the form as: *{domain_id}.{scenario_id}*. The `domain_id` could be one of the above listed five environment ids, and the `scenario_id` can be found in the full list of them from the documentation of pettingzoo.

```
from malib.rollout.envs.pettingzoo.scenario_configs_ref import SCENARIO_CONFIGS

for env_id, scenario_configs in SCENARIO_CONFIGS.items():
    env = PettingZooEnv(env_id=env_id, scenario_configs=scenario_configs)
```

(continues on next page)

(continued from previous page)

```
    action_spaces = env.action_spaces

    _, observations = env.reset()
    done = False

    while not done:
        actions = {k: action_spaces[k].sample() for k in observations.keys()}
        _, observations, rewards, dones, infos = env.step(actions)
        done = dones["__all__"]
```

As pettingzoo supports two simulation modes, i.e., `AECEnv` and `ParallelEnv`, users can switch it with specifying `parallel_simulate` in `scenario_configs`. `True` for `ParallelEnv`, and `False` for `AECEnv`.

5.1.5 SMAC: StarCraftII

coming soon ...

5.1.6 Google Research Football

coming soon ...

5.2 Environment Customiztion

MALib defines a specific class of `Environment` which is similar to `gym.Env` with some modifications to support multi-agent scenarios.

5.2.1 Customization

Interaction interfaces, e.g., `step` and `reset`, take a dictionary as input/output type in the form of `<AgentID, content>` pairs to inform MALib of different agents' states and actions and rewards, etc. To imeplement a customized environment, some interfaces you must implement including

- `Environment.possible_agents`: a property, returns a list of enviornment agent ids.
- `Environment.observation_spaces`: a property, returns a dict of agent observation spaces.
- `Environment.action_spaces`: a property, returns a dict of agent action spaces.
- `Environment.time_step`: accept a dict of agent actions, main stepping logic function, you should implement the main loop here, then the `Environment.step` function will analyze its return and record time stepping information as follows:

```
def step(
    self, actions: Dict[AgentID, Any]
) -> Tuple[
    Dict[AgentID, Any],
    Dict[AgentID, Any],
    Dict[AgentID, float],
    Dict[AgentID, bool],
    Any,
```

(continues on next page)

(continued from previous page)

```
]:
    """Return a 5-tuple as (state, observation, reward, done, info). Each␣
↪item is a dict maps from agent id to entity.

    Note:
        If state return of this environment is not activated, the return␣
↪state would be None.

    Args:
        actions (Dict[AgentID, Any]): A dict of agent actions.

    Returns:
        Tuple[ Dict[AgentID, Any], Dict[AgentID, Any], Dict[AgentID, float],
↪ Dict[AgentID, bool], Any]: A tuple follows the order as (state,␣
↪observation, reward, done, info).
    """

    self.cnt += 1
    rets = list(self.time_step(actions))
    rets[3]["__all__"] = self.env_done_check(rets[3])
    if rets[3]["__all__"]:
        rets[3] = {k: True for k in rets[3].keys()}
    rets = tuple(rets)
    self.record_episode_info_step(*rets)
    # state, obs, reward, done, info.
    return rets
```

MALib also supports *Wrapper* functionality and provides a *GroupWrapper* to map agent id to some group id.

5.2.2 Vectorization

MALib supports interacting with multiple environments in parallel with the implementation of auto-vectorized environment interface implemented in 'malib.rollout.env.vector_env'.

For users who want to use parallel rollout, he/she needs to modify certain contents in *rollout_config*.

```
rollout_config = {
    "fragment_length": 2000,  # every thread
    "max_step": 200,
    "num_eval_episodes": 10,
    "num_threads": 2,
    "num_env_per_thread": 10,
    "num_eval_threads": 1,
    "use_subproc_env": False,
    "batch_mode": "time_step",
    "postprocessor_types": ["defaults"],
    # every # rollout epoch run evaluation.
    "eval_interval": 1,
    "inference_server": "ray",  # three kinds of inference server: `local`, `pipe` and␣
↪`ray`
}
```

CHAPTER

SIX

CLUSTERED DEPLOYMENT

This page introduce the method to set up and deploy your training on a Ray cluster. We provide a manually way to do that. Users can also refer the documentation on ray-project to get other ways such as kubernates deployment.

Note: Please make sure you have installed the MALib on machines that you will use as cluster nodes, and the project path should be the same for all of them.

This section assumes that you have a list of machines and that the nodes in the cluster share the same network. It also assumes that Ray is installed on each machine. You can use pip to install the ray command line tool with cluster launcher support. Follow the Ray installation instructions for more details.

6.1 Start the Head Node

Choose any node to be the head node and run the following. If the `--port` argument is omitted, Ray will first choose port 6379, and then fall back to a random port if in 6379 is in use.

```
# start head of ray cluster at 6379, and the monitor at 8265
ray start --head --port=6379 --dashboard-port=8265
```

The command will print out the Ray cluster address, which can be passed to ray start on other machines to start the worker nodes (see below). If you receive a ConnectionError, check your firewall settings and network configuration.

6.2 Start Worker Nodes

Then on each of the other nodes, run the following command to connect to the head node you just created.

```
ray start --address=<head-node-address:port>
```

Make sure to replace `head-node-address:port` with the value printed by the command on the head node (it should look something like 123.45.67.89:6379).

Note that if your compute nodes are on their own subnetwork with Network Address Translation, the address printed by the head node will not work if connecting from a machine outside that subnetwork. You will need to use a head node address reachable from the remote machine. If the head node has a domain address like compute04.berkeley.edu, you can simply use that in place of an IP address and rely on DNS.

Ray auto-detects the resources (e.g., CPU) available on each node, but you can also manually override this by passing custom resources to the `ray start` command. For example, if you wish to specify that a machine has 10 CPUs

and 1 GPU available for use by Ray, you can do this with the flags `--num-cpus=10` and `--num-gpus=1`. See the Configuration page for more information.

Check your running task return the cluster resources info as correct as display two nodes here:

```
[2022-11-19 19:46:24,060][INFO] (runner:81) Ray cluster resources info: {'memory':␣
↪361143336347.0, 'accelerator_type:G': 1.0, 'object_store_memory': 159061429861.0,
↪'node:192.168.2.111': 1.0, 'GPU': 3.0, 'CPU': 320.0, 'accelerator_type:RTX': 1.0,
↪'node:192.168.2.54': 1.0}
```

and then the running logs will be printed in the head node as follows:

```
(PBRolloutWorker pid=1877064) [2022-11-19 19:47:17,211][INFO] (rolloutworker:458)␣
↪Evaluation at epoch: 0
(PBRolloutWorker pid=1877064) {'agent_reward/agent_max': 11.0,
(PBRolloutWorker pid=1877064)  'agent_reward/agent_mean': 9.338095238095239,
(PBRolloutWorker pid=1877064)  'agent_reward/agent_min': 5.0,
(PBRolloutWorker pid=1877064)  'agent_step/agent_max': 11.0,
(PBRolloutWorker pid=1877064)  'agent_step/agent_mean': 9.338095238095239,
(PBRolloutWorker pid=1877064)  'agent_step/agent_min': 5.0,
(PBRolloutWorker pid=1877064)  'env_step_max': 11,
(PBRolloutWorker pid=1877064)  'env_step_mean': 9.338095238095239,
(PBRolloutWorker pid=1877064)  'env_step_min': 5,
(PBRolloutWorker pid=1877064)  'episode_reward_max': 11.0,
(PBRolloutWorker pid=1877064)  'episode_reward_mean': 9.338095238095239,
(PBRolloutWorker pid=1877064)  'episode_reward_min': 5.0,
(PBRolloutWorker pid=1877064)  'performance': {'ave_rollout_FPS': 1131.3788088578215,
(PBRolloutWorker pid=1877064)                  'rollout_FPS': 1131.3788088578215,
(PBRolloutWorker pid=1877064)                  'rollout_iter_rate': 0.0327749285686886}}
(PBRolloutWorker pid=1877064) [2022-11-19 19:47:23,134][INFO] (rolloutworker:458)␣
↪Evaluation at epoch: 1
(PBRolloutWorker pid=1877064) {'agent_reward/agent_max': 22.0,
(PBRolloutWorker pid=1877064)  'agent_reward/agent_mean': 9.625615763546797,
(PBRolloutWorker pid=1877064)  'agent_reward/agent_min': 2.0,
(PBRolloutWorker pid=1877064)  'agent_step/agent_max': 22.0,
(PBRolloutWorker pid=1877064)  'agent_step/agent_mean': 9.625615763546797,
(PBRolloutWorker pid=1877064)  'agent_step/agent_min': 2.0,
(PBRolloutWorker pid=1877064)  'env_step_max': 22,
(PBRolloutWorker pid=1877064)  'env_step_mean': 9.625615763546797,
(PBRolloutWorker pid=1877064)  'env_step_min': 2,
(PBRolloutWorker pid=1877064)  'episode_reward_max': 22.0,
(PBRolloutWorker pid=1877064)  'episode_reward_mean': 9.625615763546797,
(PBRolloutWorker pid=1877064)  'episode_reward_min': 2.0,
(PBRolloutWorker pid=1877064)  'performance': {'ave_rollout_FPS': 1414.794048720742,
(PBRolloutWorker pid=1877064)                  'rollout_FPS': 1698.2092885836623,
(PBRolloutWorker pid=1877064)                  'rollout_iter_rate': 0.05489372662924034}}
```

6.3 Dashboard

As you've start a dashboard at port 8265, you can see the monitor resources as

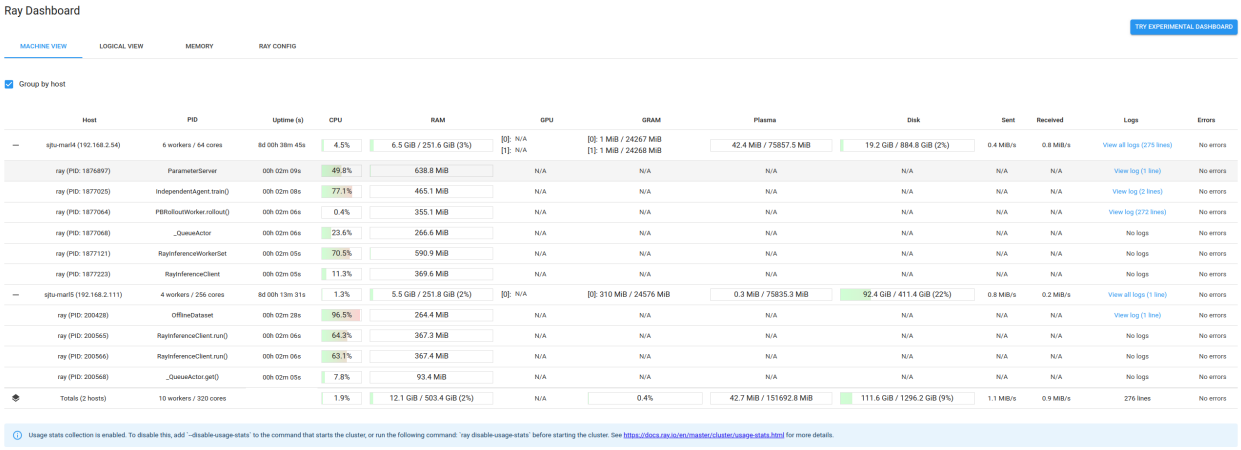

Fig. 1: Ray cluster monitor

CHAPTER

SEVEN

MALIB.AGENT PACKAGE

7.1 Submodules

7.2 malib.agent.agent_interface module

class malib.agent.agent_interface.**AgentInterface**(*experiment_tag: str*, *runtime_id: str*, *log_dir: str*, *env_desc: Dict[str, Any]*, *algorithms: Dict[str, Tuple[Type, Type, Dict, Dict]]*, *agent_mapping_func: Callable[[str], str]*, *governed_agents: Tuple[str]*, *trainer_config: Dict[str, Any]*, *custom_config: Optional[Dict[str, Any]] = None*, *local_buffer_config: Optional[Dict] = None*, *verbose: bool = True*)

Bases: *RemoteInterface*, ABC

Base class of agent interface, for training

Construct agent interface for training.

Parameters

- **experiment_tag** (*str*) – Experiment tag.
- **runtime_id** (*str*) – Assigned runtime id, should be an element of the agent mapping results.
- **log_dir** (*str*) – The directory for logging.
- **env_desc** (*Dict[str, Any]*) – A dict that describes the environment property.
- **algorithms** (*Dict[str, Tuple[Type, Type, Dict]]*) – A dict that describes the algorithm candidates. Each is a tuple of *policy_cls*, *trainer_cls*, *model_config* and *custom_config*.
- **agent_mapping_func** (*Callable[[AgentID], str]*) – A function that defines the rule of agent groupping.
- **governed_agents** (*Tuple[AgentID]*) – A tuple that records which agents is related to this training procedures. Note that it should be a subset of the original set of environment agents.
- **trainer_config** (*Dict[str, Any]*) – Trainer configuration.
- **custom_config** (*Dict[str, Any], optional*) – A dict of custom configuration. Defaults to None.

- **local_buffer_config** (*Dict, optional*) – A dict for local buffer configuration. Defaults to None.
- **verbose** (*bool, True*) – Enable logging or not. Defaults to True.

add_policies(*n: int*) → *StrategySpec*

Construct *n* new policies and return the latest strategy spec.

Parameters
n (*int*) – Indicates how many new policies will be added.

Returns
The latest strategy spec instance.

Return type
StrategySpec

connect(*max_tries: int = 10*, *dataset_server_ref: Optional[str] = None*, *parameter_server_ref: Optional[str] = None*)

Try to connect with backend, i.e., parameter server and offline dataset server. If the reference of dataset server or parameter server is not been given, then the agent will use default settings.

Parameters

- **max_tries** (*int, optional*) – Maximum of trails. Defaults to 10.
- **dataset_server_ref** (*str, optional*) – Name of ray-based dataset server. Defaults to None.
- **parameter_server_ref** (*str, optional*) – Name of ray-based parameter server. Defaults to None.

property device: Union[str, DeviceObjType]

Retrive device name.

Returns
Device name.

Return type
Union[str, torch.DeviceObjType]

get_algorithm(*key: str*) → Any

Return a copy of algorithm configuration with given key, if not exist, raise KeyError.

Parameters
key (*str*) – Algorithm configuration reference key.

Raises
KeyError – No such an algorithm configuration relates to the give key.

Returns
Algorithm configuration, mabe a dict.

Return type
Any

get_algorthms() → Dict[str, Any]

Return a copy of full algorithm configurations.

Returns
Full algorithm configurations.

Return type
Dict[str, Any]

`get_interface_state`() → Dict[str, Any]

Return a dict that describes the current learning state.

Returns
A dict of learning state.

Return type
Dict[str, Any]

`property governed_agents: Tuple[str]`

Return a tuple of governed environment agents.

Returns
A tuple of agent ids.

Return type
Tuple[str]

`abstract multiagent_post_process`(*batch_info: Union[Dict[str, Tuple[*Batch*, List[int]]], Tuple[*Batch*, List[int]]]*) → Dict[str, Any]

Merge agent buffer here and return the merged buffer.

Parameters
batch_info (*Union[Dict[AgentID, Tuple[*Batch*, List[int]]], Tuple[*Batch*, List[int]]]*) – Batch info, could be a dict of agent batch info or a tuple.

Returns
A merged buffer dict.

Return type
Dict[str, Any]

`pull`()

Pull remote weights to update local version.

`push`()

Push local weights to remote server

`reset`()

Reset training state.

`sync_remote_parameters`()

Push latest network parameters of active policies to remote parameter server.

`train`(*data_request_identifier: str*, *reset_state: bool = True*) → Dict[str, Any]

Executes training task and returns the final interface state.

Parameters

- **stopping_conditions** (*Dict[str, Any]*) – Control the training stepping.
- **reset_tate** (*bool, optional*) – Reset interface state or not. Default is True.

Returns
A dict that describes the final state.

Return type
Dict[str, Any]

7.3 malib.agent.async_agent module

class malib.agent.async_agent.**AsyncAgent**(*experiment_tag: str*, *runtime_id: str*, *log_dir: str*, *env_desc: Dict[str, Any]*, *algorithms: Dict[str, Tuple[Type, Type, Dict, Dict]]*, *agent_mapping_func: Callable[[str], str]*, *governed_agents: Tuple[str]*, *trainer_config: Dict[str, Any]*, *custom_config: Optional[Dict[str, Any]] = None*, *local_buffer_config: Optional[Dict] = None*, *verbose: bool = True*)

Bases: *AgentInterface*

Construct agent interface for training.

Parameters

- **experiment_tag** (*str*) – Experiment tag.
- **runtime_id** (*str*) – Assigned runtime id, should be an element of the agent mapping results.
- **log_dir** (*str*) – The directory for logging.
- **env_desc** (*Dict[str, Any]*) – A dict that describes the environment property.
- **algorithms** (*Dict[str, Tuple[Type, Type, Dict]]*) – A dict that describes the algorithm candidates. Each is a tuple of *policy_cls*, *trainer_cls*, *model_config* and *custom_config*.
- **agent_mapping_func** (*Callable[[AgentID], str]*) – A function that defines the rule of agent groupping.
- **governed_agents** (*Tuple[AgentID]*) – A tuple that records which agents is related to this training procedures. Note that it should be a subset of the original set of environment agents.
- **trainer_config** (*Dict[str, Any]*) – Trainer configuration.
- **custom_config** (*Dict[str, Any], optional*) – A dict of custom configuration. Defaults to None.
- **local_buffer_config** (*Dict, optional*) – A dict for local buffer configuration. Defaults to None.
- **verbose** (*bool, True*) – Enable logging or not. Defaults to True.

7.4 malib.agent.indepdent_agent module

class malib.agent.indepdent_agent.**IndependentAgent**(*experiment_tag: str*, *runtime_id: str*, *log_dir: str*, *env_desc: Dict[str, Any]*, *algorithms: Dict[str, Tuple[Dict, Dict, Dict]]*, *agent_mapping_func: Callable[[str], str]*, *governed_agents: Tuple[str]*, *trainer_config: Dict[str, Any]*, *custom_config: Optional[Dict[str, Any]] = None*, *local_buffer_config: Optional[Dict] = None*, *verbose: bool = True*)

Bases: *AgentInterface*

Construct agent interface for training.

Parameters

- **experiment_tag** (*str*) – Experiment tag.
- **runtime_id** (*str*) – Assigned runtime id, should be an element of the agent mapping results.
- **log_dir** (*str*) – The directory for logging.
- **env_desc** (*Dict[str, Any]*) – A dict that describes the environment property.
- **algorithms** (*Dict[str, Tuple[Type, Type, Dict]]*) – A dict that describes the algorithm candidates. Each is a tuple of *policy_cls*, *trainer_cls*, *model_config* and *custom_config*.
- **agent_mapping_func** (*Callable[[AgentID], str]*) – A function that defines the rule of agent groupping.
- **governed_agents** (*Tuple[AgentID]*) – A tuple that records which agents is related to this training procedures. Note that it should be a subset of the original set of environment agents.
- **trainer_config** (*Dict[str, Any]*) – Trainer configuration.
- **custom_config** (*Dict[str, Any], optional*) – A dict of custom configuration. Defaults to None.
- **local_buffer_config** (*Dict, optional*) – A dict for local buffer configuration. Defaults to None.
- **verbose** (*bool, True*) – Enable logging or not. Defaults to True.

multiagent_post_process(*batch_info: Union[Dict[str, Tuple[Batch, List[int]]], Tuple[Batch, List[int]]]*) → Dict[str, Any]

Merge agent buffer here and return the merged buffer.

Parameters
batch_info (*Union[Dict[AgentID, Tuple[Batch, List[int]]], Tuple[Batch, List[int]]]*) – Batch info, could be a dict of agent batch info or a tuple.

Returns
A merged buffer dict.

Return type
Dict[str, Any]

7.5 malib.agent.manager module

class malib.agent.manager.**TrainingManager**(*experiment_tag: str*, *stopping_conditions: Dict[str, Any]*, *algorithms: Dict[str, Any]*, *env_desc: Dict[str, Any]*, *agent_mapping_func: Callable[[str], str]*, *training_config: Dict[str, Any]*, *log_dir: str*, *remote_mode: bool = True*, *resource_config: Optional[Dict[str, Any]] = None*, *verbose: bool = True*)

Bases: *Manager*

Create an TrainingManager instance which is responsible for the multi agent training tasks execution and rollout task requests sending.

Parameters

- **experiment_tag** (*str*) – Experiment identifier, for data tracking.
- **algorithms** (*Dict[str, Any]*) – The algorithms configuration candidates.
- **env_desc** (*Dict[str, Any]*) – The description for environment generation.
- **interface_config** (*Dict[str, Any]*) – Configuration for agent training inferece construction, keys include *type* and *custom_config*, a dict.
- **agent_mapping_func** (*Callable[[AgentID], str]*) – The mapping function maps agent id to training interface id.
- **training_config** (*Dict[str, Any]*) – Training configuration, for agent interface, keys include *type*, *trainer_config* and *custom_config*.
- **log_dir** (*str*) – Directory for logging.
- **remote_mode** (*bool, Optional*) – Init agent interfaces as remote actor or not. Default is True.

add_policies(*interface_ids: Optional[Sequence[str]] = None*, *n: Union[int, Dict[str, int]] = 1*) → Dict[str, Type[*StrategySpec*]]

Notify interface *interface_id* add *n* policies and return the newest strategy spec.

Parameters

- **interface_ids** (*Sequence[str]*) – Registered agent interface id.
- **n** (*int, optional*) – Indicates how many policies will be added.

Returns
A dict of strategy specs, maps from runtime ids to strategy specs.

Return type
Dict[str, Type[*StrategySpec*]]

property agent_groups: Dict[str, Set[str]]

A dict describes the agent grouping, maps from runtime ids to agent sets.

Returns
A dict of agent set.

Return type
Dict[str, Set[AgentID]]

get_exp(*policy_distribution*)

Compute exploitability

retrive_results()

run(*data_request_identifiers: Dict[str, str]*)

Start training thread without blocking

property runtime_ids: Tuple[str]

terminate() → None

Terminate all training actors.

property workers: List[*RemoteInterface*]

7.6 malib.agent.team_agent module

class malib.agent.team_agent.**TeamAgent**(*experiment_tag: str*, *runtime_id: str*, *log_dir: str*, *env_desc: Dict[str, Any]*, *algorithms: Dict[str, Tuple[Type, Type, Dict, Dict]]*, *agent_mapping_func: Callable[[str], str]*, *governed_agents: Tuple[str]*, *trainer_config: Dict[str, Any]*, *custom_config: Optional[Dict[str, Any]] = None*, *local_buffer_config: Optional[Dict] = None*, *verbose: bool = True*)

Bases: *AgentInterface*

Construct agent interface for training.

Parameters

- **experiment_tag** (*str*) – Experiment tag.
- **runtime_id** (*str*) – Assigned runtime id, should be an element of the agent mapping results.
- **log_dir** (*str*) – The directory for logging.
- **env_desc** (*Dict[str, Any]*) – A dict that describes the environment property.
- **algorithms** (*Dict[str, Tuple[Type, Type, Dict]]*) – A dict that describes the algorithm candidates. Each is a tuple of *policy_cls*, *trainer_cls*, *model_config* and *custom_config*.
- **agent_mapping_func** (*Callable[[AgentID], str]*) – A function that defines the rule of agent groupping.
- **governed_agents** (*Tuple[AgentID]*) – A tuple that records which agents is related to this training procedures. Note that it should be a subset of the original set of environment agents.
- **trainer_config** (*Dict[str, Any]*) – Trainer configuration.
- **custom_config** (*Dict[str, Any], optional*) – A dict of custom configuration. Defaults to None.
- **local_buffer_config** (*Dict, optional*) – A dict for local buffer configuration. Defaults to None.
- **verbose** (*bool, True*) – Enable logging or not. Defaults to True.

multiagent_post_process(*batch_info: Union[Dict[str, Tuple[Batch, List[int]]], Tuple[Batch, List[int]]]*) → Dict[str, *Batch*]

Merge agent buffer here and return the merged buffer.

Parameters

batch_info (*Union[Dict[AgentID, Tuple[Batch, List[int]]], Tuple[Batch, List[int]]]*) – Batch info, could be a dict of agent batch info or a tuple.

Returns

A merged buffer dict.

Return type

Dict[str, Any]

CHAPTER

EIGHT

MALIB.BACKEND PACKAGE

8.1 Submodules

8.2 malib.backend.offline_dataset_server module

class malib.backend.offline_dataset_server.**OfflineDataset**(*table_capacity: int*, *max_consumer_size: int = 1024*)

Bases: *RemoteInterface*

Construct an offline datataset. It maintans a dict of datatable, each for a training instance.

Parameters

- **table_capacity** (*int*) – Table capacity, it indicates the buffer size of each data table.
- **max_consumer_size** (*int, optional*) – Defines the maximum of concurrency. Defaults to 1024.

end_consumer_pipe(*name: str*)

Kill a consumer pipeline with given table name.

Parameters
name (*str*) – Name of related datatable.

end_producer_pipe(*name: str*)

Kill a producer pipe with given name.

Parameters
name (*str*) – The name of related data table.

start()

start_consumer_pipe(*name: str*, *batch_size: int*) → Tuple[str, Queue]

Start a consumer pipeline, if there is no such a table that named as *name*, the function will be stucked until the table has been created.

Parameters

- **name** (*str*) – Name of datatable.
- **batch_size** (*int*) – Batch size.

Returns
A tuple of table name and queue for retrieving samples.

Return type
Tuple[str, Queue]

start_producer_pipe(*name: str*, *stack_num: int = 1*, *ignore_obs_next: bool = False*, *save_only_last_obs: bool = False*, *sample_avail: bool = False*, ***kwargs*) → Tuple[str, Queue]

Start a producer pipeline and create a datatable if not exisits.

Parameters

- **name** (*str*) – The name of datatable need to access
- **stack_num** (*int, optional*) – Indicates how many steps are stacked in a single data sample. Defaults to 1.
- **ignore_obs_next** (*bool, optional*) – Ignore the next observation or not. Defaults to False.
- **save_only_last_obs** (*bool, optional*) – Either save only the last observation frame. Defaults to False.
- **sample_avail** (*bool, optional*) – Sample action maks or not. Defaults to False.

Returns
A tuple of table name and queue for insert samples.

Return type
Tuple[str, Queue]

malib.backend.offline_dataset_server.**read_table**(*marker: RWLockFair*, *buffer: Union[MultiagentReplayBuffer, ReplayBuffer]*, *batch_size: int*, *reader: Queue*)

malib.backend.offline_dataset_server.**write_table**(*marker: RWLockFair*, *buffer: Union[MultiagentReplayBuffer, ReplayBuffer]*, *writer: Queue*)

8.3 malib.backend.parameter_server module

class malib.backend.parameter_server.**ParameterServer**(***kwargs*)

Bases: *RemoteInterface*

apply_gradients(*table_name: str*, *gradients: Sequence[Any]*)

Apply gradients to a data table.

Parameters

- **table_name** (*str*) – The specified table name.
- **gradients** (*Sequence[Any]*) – Given gradients to update parameters.

Raises
NotImplementedError – Not implemented yet.

create_table(*strategy_spec:* StrategySpec) → str

Create parameter table with given strategy spec. This function will traverse existing policy id in this spec, then generate table for policy ids which have no cooresponding tables.

Parameters
strategy_spec (StrategySpec) – A startegy spec instance.

Returns
Table name formatted as *{startegy_spec_id}/{policy_id}*.

Return type
str

`get_weights`(*spec_id: str*, *spec_policy_id: str*) → Dict[str, Any]

Request for weight retrive, return a dict includes keys: *spec_id*, *spec_policy_id* and *weights*.

Parameters

- **spec_id** (*str*) – Strategy spec id.
- **spec_policy_id** (*str*) – Related policy id.

Returns
A dict.

Return type
Dict[str, Any]

`set_weights`(*spec_id: str*, *spec_policy_id: str*, *state_dict: Dict[str, Any]*)

Set weights to a parameter table. The table name will be defined as *{spec_id}/{spec_policy_id}*

Parameters

- **spec_id** (*str*) – StrategySpec id.
- **spec_policy_id** (*str*) – Policy id in the specified strategy spec.
- **state_dict** (*Dict[str, Any]*) – A dict that specify the parameters.

`start`()

For debug

class malib.backend.parameter_server.**Table**(*policy_meta_data: Dict[str, Any]*)

Bases: object

`apply_gradients`(**gradients*)

`get_weights`() → Dict[str, Any]

Retrive model weights.

Returns
Weights dict

Return type
Dict[str, Any]

`set_weights`(*state_dict: Dict[str, Any]*)

Update weights with given weights.

Parameters
state_dict (*Dict[str, Any]*) – A dict of weights

CHAPTER

NINE

MALIB.COMMON PACKAGE

9.1 Submodules

9.2 malib.common.distributions module

Probability distributions. Reference: https://github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3/common/distributions.py

class malib.common.distributions.**BernoulliDistribution**(*action_dims: int*)

Bases: *Distribution*

Bernoulli distribution for MultiBinary action spaces.

Parameters
action_dim – Number of binary actions

actions_from_params(*action_logits: Tensor*, *deterministic: bool = False*) → Tensor

Returns samples from the probability distribution given its parameters.

Returns
actions

entropy() → Tensor

Returns Shannon's entropy of the probability

Returns
the entropy, or None if no analytical form is known

log_prob(*actions: Tensor*) → Tensor

Returns the log likelihood

Parameters
x – the taken action

Returns
The log likelihood of the distribution

log_prob_from_params(*action_logits: Tensor*) → Tuple[Tensor, Tensor]

Returns samples and the associated log probabilities from the probability distribution given its parameters.

Returns
actions and log prob

mode() → Tensor

Returns the most likely action (deterministic output) from the probability distribution

Returns
the stochastic action

proba_distribution(*action_logits: Tensor*) → *BernoulliDistribution*

Set parameters of the distribution.

Returns
self

proba_distribution_net(*latent_dim: int*) → Module

Create the layer that represents the distribution: it will be the logits of the Bernoulli distribution.

Parameters
latent_dim – Dimension of the last layer of the policy network (before the action layer)

Returns

sample() → Tensor

Returns a sample from the probability distribution

Returns
the stochastic action

class malib.common.distributions.**CategoricalDistribution**(*action_dim: int*)

Bases: *Distribution*

Categorical distribution for discrete actions.

Parameters
action_dim (*int*) – Number of discrete actions.

actions_from_params(*action_logits: Tensor*, *deterministic: bool = False*) → Tensor

Returns samples from the probability distribution given its parameters.

Returns
actions

entropy() → Tensor

Returns Shannon's entropy of the probability

Returns
the entropy, or None if no analytical form is known

log_prob(*actions: Tensor*) → Tensor

Returns the log likelihood

Parameters
x – the taken action

Returns
The log likelihood of the distribution

log_prob_from_params(*action_logits: Tensor*, *deterministic: bool = False*) → Tuple[Tensor, Tensor]

Returns samples and the associated log probabilities from the probability distribution given its parameters.

Returns
actions and log prob

mode() → Tensor

Returns the most likely action (deterministic output) from the probability distribution

Returns
the stochastic action

prob() → Tensor

Return a tensor which indicates the distribution

Returns
A distribution tensor

Return type
torch.Tensor

proba_distribution(*action_logits: Tensor*, *action_mask: Optional[Tensor] = None*) → *CategoricalDistribution*

Set parameters of the distribution.

Returns
self

proba_distribution_net(*latent_dim: int*) → Module

Create the layer that represents the distribution: it will be the logits of the Categorical distribution. You can then get probabilities using a softmax.

Parameters
latent_dim – Dimension of the last layer of the policy network (before the action layer)

Returns

sample() → Tensor

Returns a sample from the probability distribution

Returns
the stochastic action

class malib.common.distributions.**DiagGaussianDistribution**(*action_dim: int*)

Bases: *Distribution*

Gaussian distribution with diagonal covariance matrix, for continuous actions.

Parameters
action_dim – Dimension of the action space.

actions_from_params(*mean_actions: Tensor*, *log_std: Tensor*, *deterministic: bool = False*) → Tensor

Returns samples from the probability distribution given its parameters.

Returns
actions

entropy() → Tensor

Returns Shannon's entropy of the probability

Returns
the entropy, or None if no analytical form is known

log_prob(*actions: Tensor*) → Tensor

Get the log probabilities of actions according to the distribution. Note that you must first call the proba_distribution() method.

Parameters
actions –

Returns

log_prob_from_params(*mean_actions: Tensor*, *log_std: Tensor*) → Tuple[Tensor, Tensor]

Compute the log probability of taking an action given the distribution parameters.

Parameters

- **mean_actions** –
- **log_std** –

Returns

mode() → Tensor

Returns the most likely action (deterministic output) from the probability distribution

Returns
the stochastic action

prob() → Tensor

Return a tensor which indicates the distribution

Returns
A distribution tensor

Return type
torch.Tensor

proba_distribution(*mean_actions: Tensor*, *log_std: Tensor*) → *DiagGaussianDistribution*

Create the distribution given its parameters (mean, std)

Parameters

- **mean_actions** –
- **log_std** –

Returns

proba_distribution_net(*latent_dim: int*, *log_std_init: float = 0.0*) → Tuple[Module, Parameter]

Create the layers and parameter that represent the distribution: one output will be the mean of the Gaussian, the other parameter will be the standard deviation (log std in fact to allow negative values)

Parameters

- **latent_dim** – Dimension of the last layer of the policy (before the action layer)
- **log_std_init** – Initial value for the log standard deviation

Returns

sample() → Tensor

Returns a sample from the probability distribution

Returns
the stochastic action

class malib.common.distributions.**Distribution**

Bases: ABC

Abstract base class for distributions.

abstract actions_from_params(**args*, ***kwargs*) → Tensor

Returns samples from the probability distribution given its parameters.

Returns
actions

abstract entropy() → Optional[Tensor]

Returns Shannon's entropy of the probability

Returns
the entropy, or None if no analytical form is known

get_actions(*deterministic: bool = False*) → Tensor

Return actions according to the probability distribution.

Parameters
deterministic –

Returns

abstract log_prob(*x: Tensor*) → Tensor

Returns the log likelihood

Parameters
x – the taken action

Returns
The log likelihood of the distribution

abstract log_prob_from_params(**args*, ***kwargs*) → Tuple[Tensor, Tensor]

Returns samples and the associated log probabilities from the probability distribution given its parameters.

Returns
actions and log prob

abstract mode() → Tensor

Returns the most likely action (deterministic output) from the probability distribution

Returns
the stochastic action

abstract prob() → Tensor

Return a tensor which indicates the distribution

Returns
A distribution tensor

Return type
torch.Tensor

abstract proba_distribution(**args*, ***kwargs*) → *Distribution*

Set parameters of the distribution.

Returns
self

abstract proba_distribution_net(**args*, ***kwargs*) → Union[Module, Tuple[Module, Parameter]]

Create the layers and parameters that represent the distribution.

Subclasses must define this, but the arguments and return type vary between concrete classes.

abstract sample() → Tensor

Returns a sample from the probability distribution

Returns
the stochastic action

class malib.common.distributions.**MaskedCategorical**(*scores*, *mask=None*)

Bases: object

property entropy

log_prob(*value*)

property logits

static masked_softmax(*logits*, *mask*)

This method will return valid probability distribution for the particular instance if its corresponding row in the *mask* matrix is not a zero vector. Otherwise, a uniform distribution will be returned. This is just a technical workaround that allows *Categorical* class usage. If probs doesn't sum to one there will be an exception during sampling.

property normalized_entropy

property probs

rsample(*temperature=None*, *gumbel_noise=None*)

sample()

class malib.common.distributions.**MultiCategoricalDistribution**(*action_dims: List[int]*)

Bases: *Distribution*

MultiCategorical distribution for multi discrete actions.

Parameters
action_dims – List of sizes of discrete action spaces

actions_from_params(*action_logits: Tensor*, *deterministic: bool = False*) → Tensor

Returns samples from the probability distribution given its parameters.

Returns
actions

entropy() → Tensor

Returns Shannon's entropy of the probability

Returns
the entropy, or None if no analytical form is known

log_prob(*actions: Tensor*) → Tensor

Returns the log likelihood

Parameters
x – the taken action

Returns
The log likelihood of the distribution

log_prob_from_params(*action_logits: Tensor*) → Tuple[Tensor, Tensor]

Returns samples and the associated log probabilities from the probability distribution given its parameters.

Returns
actions and log prob

mode() → Tensor

Returns the most likely action (deterministic output) from the probability distribution

Returns
the stochastic action

proba_distribution(*action_logits: Tensor*) → *MultiCategoricalDistribution*

Set parameters of the distribution.

Returns
self

proba_distribution_net(*latent_dim: int*) → Module

Create the layer that represents the distribution: it will be the logits (flattened) of the MultiCategorical distribution. You can then get probabilities using a softmax on each sub-space.

Parameters
latent_dim – Dimension of the last layer of the policy network (before the action layer)

Returns

sample() → Tensor

Returns a sample from the probability distribution

Returns
the stochastic action

class malib.common.distributions.**SquashedDiagGaussianDistribution**(*action_dim: int*, *epsilon: float = 1e-06*)

Bases: *DiagGaussianDistribution*

Gaussian distribution with diagonal covariance matrix, followed by a squashing function (tanh) to ensure bounds.

Parameters

- **action_dim** – Dimension of the action space.
- **epsilon** – small value to avoid NaN due to numerical imprecision.

entropy() → Optional[Tensor]

Returns Shannon's entropy of the probability

Returns
the entropy, or None if no analytical form is known

log_prob(*actions: Tensor*, *gaussian_actions: Optional[Tensor] = None*) → Tensor

Get the log probabilities of actions according to the distribution. Note that you must first call the proba_distribution() method.

Parameters
actions –

Returns

log_prob_from_params(*mean_actions: Tensor*, *log_std: Tensor*) → Tuple[Tensor, Tensor]

Compute the log probability of taking an action given the distribution parameters.

Parameters

- **mean_actions** –
- **log_std** –

Returns

mode() → Tensor

Returns the most likely action (deterministic output) from the probability distribution

Returns
the stochastic action

proba_distribution(*mean_actions: Tensor*, *log_std: Tensor*) → *SquashedDiagGaussianDistribution*

Create the distribution given its parameters (mean, std)

Parameters

- **mean_actions** –
- **log_std** –

Returns

sample() → Tensor

Returns a sample from the probability distribution

Returns
the stochastic action

class malib.common.distributions.**StateDependentNoiseDistribution**(*action_dim: int*, *full_std: bool = True*, *use_expln: bool = False*, *squash_output: bool = False*, *learn_features: bool = False*, *epsilon: float = 1e-06*)

Bases: *Distribution*

Distribution class for using generalized State Dependent Exploration (gSDE). Paper: https://arxiv.org/abs/2005.05719

It is used to create the noise exploration matrix and compute the log probability of an action with that noise.

Parameters

- **action_dim** – Dimension of the action space.
- **full_std** – Whether to use (n_features x n_actions) parameters for the std instead of only (n_features,)
- **use_expln** – Use `expln()` function instead of `exp()` to ensure a positive standard deviation (cf paper). It allows to keep variance above zero and prevent it from growing too fast. In practice, `exp()` is usually enough.
- **squash_output** – Whether to squash the output using a tanh function, this ensures bounds are satisfied.
- **learn_features** – Whether to learn features for gSDE or not. This will enable gradients to be backpropagated through the features `latent_sde` in the code.
- **epsilon** – small value to avoid NaN due to numerical imprecision.

actions_from_params(*mean_actions: Tensor*, *log_std: Tensor*, *latent_sde: Tensor*, *deterministic: bool = False*) → Tensor

Returns samples from the probability distribution given its parameters.

Returns
actions

entropy() → Optional[Tensor]

Returns Shannon's entropy of the probability

Returns
the entropy, or None if no analytical form is known

get_noise(*latent_sde: Tensor*) → Tensor

get_std(*log_std: Tensor*) → Tensor

Get the standard deviation from the learned parameter (log of it by default). This ensures that the std is positive.

Parameters
log_std –

Returns

log_prob(*actions: Tensor*) → Tensor

Returns the log likelihood

Parameters
x – the taken action

Returns
The log likelihood of the distribution

log_prob_from_params(*mean_actions: Tensor*, *log_std: Tensor*, *latent_sde: Tensor*) → Tuple[Tensor, Tensor]

Returns samples and the associated log probabilities from the probability distribution given its parameters.

Returns
actions and log prob

mode() → Tensor

Returns the most likely action (deterministic output) from the probability distribution

Returns
the stochastic action

proba_distribution(*mean_actions: Tensor*, *log_std: Tensor*, *latent_sde: Tensor*) → *StateDependentNoiseDistribution*

Create the distribution given its parameters (mean, std)

Parameters

- **mean_actions** –
- **log_std** –
- **latent_sde** –

Returns

proba_distribution_net(*latent_dim: int*, *log_std_init: float = -2.0*, *latent_sde_dim: Optional[int] = None*) → Tuple[Module, Parameter]

Create the layers and parameter that represent the distribution: one output will be the deterministic action, the other parameter will be the standard deviation of the distribution that control the weights of the noise matrix.

Parameters

- **latent_dim** – Dimension of the last layer of the policy (before the action layer)
- **log_std_init** – Initial value for the log standard deviation
- **latent_sde_dim** – Dimension of the last layer of the features extractor for gSDE. By default, it is shared with the policy network.

Returns

sample() → Tensor

Returns a sample from the probability distribution

Returns
the stochastic action

sample_weights(*log_std: Tensor*, *batch_size: int = 1*) → None

Sample weights for the noise exploration matrix, using a centered Gaussian distribution.

Parameters

- **log_std** –
- **batch_size** –

class malib.common.distributions.**TanhBijector**(*epsilon: float = 1e-06*)

Bases: object

Bijective transformation of a probability distribution using a squashing function (tanh) TODO: use Pyro instead (https://pyro.ai/)

Parameters
epsilon – small value to avoid NaN due to numerical imprecision.

static atanh(*x: Tensor*) → Tensor

Inverse of Tanh

Taken from Pyro: https://github.com/pyro-ppl/pyro 0.5 * torch.log((1 + x) / (1 - x))

static forward(*x: Tensor*) → Tensor

static inverse(*y: Tensor*) → Tensor

Inverse tanh.

Parameters
y –

Returns

log_prob_correction(*x: Tensor*) → Tensor

malib.common.distributions.**kl_divergence**(*dist_true: Distribution*, *dist_pred: Distribution*) → Tensor

Wrapper for the PyTorch implementation of the full form KL Divergence

Parameters

- **dist_true** – the p distribution

- **dist_pred** – the q distribution

Returns
KL(dist_true||dist_pred)

malib.common.distributions.**make_proba_distribution**(*action_space: Space*, *use_sde: bool = False*, *dist_kwargs: Optional[Dict[str, Any]] = None*) → *Distribution*

Return an instance of Distribution for the correct type of action space.

Parameters

- **action_space** (*gym.spaces.Space*) – The action space.
- **use_sde** (*bool, optional*) – Force the use of StateDependentNoiseDistribution instead of DiagGaussianDistribution. Defaults to False.
- **dist_kwargs** (*Optional[Dict[str, Any]], optional*) – Keyword arguments to pass to the probability distribution. Defaults to None.

Raises
NotImplementedError – Probability distribution not implemented for the specified action space.

Returns
The appropriate Distribution object

Return type
Distribution

malib.common.distributions.**sum_independent_dims**(*tensor: Tensor*) → Tensor

Continuous actions are usually considered to be independent, so we can sum components of the `log_prob` or the entropy.

Parameters
tensor – shape: (n_batch, n_actions) or (n_batch,)

Returns
shape: (n_batch,)

9.3 malib.common.manager module

class malib.common.manager.**Manager**(*verbose: bool*)

Bases: ABC

cancel_pending_tasks()

Cancle all running tasks.

force_stop()

is_running()

retrive_results()

abstract terminate()

Resource recall

wait() → List[Any]

Wait workers to be terminated, and retrieve the executed results.

Returns
A list of results.

Return type
List[Any]

property workers: List[*RemoteInterface*]

9.4 malib.common.payoff_manager module

9.5 malib.common.strategy_spec module

class malib.common.strategy_spec.**StrategySpec**(*identifier: str*, *policy_ids: Tuple[str]*, *meta_data: Dict[str, Any]*)

Bases: object

Construct a strategy spec.

Parameters

- **identifier** (*str*) – Runtime id as identifier.
- **policy_ids** (*Tuple[PolicyID]*) – A tuple of policy id, could be empty.
- **meta_data** (*Dict[str, Any]*) – Meta data, for policy construction.

gen_policy(*device=None*) → *Policy*

Generate a policy instance with the given meta data.

Returns
A policy instance.

Return type
Policy

get_meta_data() → Dict[str, Any]

Return meta data. Keys in meta-data contains

- policy_cls: policy class type
- kwargs: a dict of parameters for policy construction
- experiment_tag: a string for experiment identification
- optim_config: optional, a dict for optimizer construction

Returns
A dict of meta data.

Return type
Dict[str, Any]

load_from_checkpoint(*policy_id: str*)

property num_policy: int

register_policy_id(*policy_id: str*)

Register new policy id, and preset prob as 0.

Parameters
policy_id (*PolicyID*) – Policy id to register.

sample() → str

Sample a policy instance. Use uniform sample if there is no presetted prob list in meta data.

Returns
A sampled policy id.

Return type
PolicyID

update_prob_list(*policy_probs: Dict[str, float]*)

Update prob list with given policy probs dict. Partial assignment is allowed.

Parameters
policy_probs (*Dict[PolicyID, float]*) – A dict that indicates which policy probs should be updated.

malib.common.strategy_spec.**validate_meta_data**(*policy_ids: Tuple[str]*, *meta_data: Dict[str, Any]*)

Validate meta data. check whether there is a valid prob list.

Parameters

- **policy_ids** (*Tuple[PolicyID]*) – A tuple of registered policy ids.
- **meta_data** (*Dict[str, Any]*) – Meta data.

CHAPTER

TEN

MALIB.MODELS PACKAGE

10.1 Subpackages

10.1.1 malib.models.torch package

Submodules

malib.models.torch.continuous module

class malib.models.torch.continuous.**Actor**(*preprocess_net: Module*, *action_shape: Sequence[int]*, *hidden_sizes: Sequence[int] = ()*, *max_action: float = 1.0*, *device: Union[str, int, device] = 'cpu'*, *preprocess_net_output_dim: Optional[int] = None*)

Bases: `Module`

Simple actor network. Will create an actor operated in continuous action space with structure of preprocess_net —> action_shape. :param preprocess_net: a self-defined preprocess_net which output a

flattened hidden state.

Parameters

- **action_shape** – a sequence of int for the shape of action.
- **hidden_sizes** – a sequence of int for constructing the MLP after preprocess_net. Default to empty sequence (where the MLP now contains only a single linear layer).
- **max_action** (*float*) – the scale for the final action logits. Default to 1.
- **preprocess_net_output_dim** (*int*) – the output dimension of preprocess_net.

For advanced usage (how to customize the network), please refer to build_the_network. .. seealso:

```
Please refer to :class:`~tianshou.utils.net.common.Net` as an instance
of how preprocess_net is suggested to be defined.
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, *state: Optional[Any] = None*, *info: Dict[str, Any] = {}*) → Tuple[Tensor, Any]

Mapping: obs -> logits -> action.

training: bool

class malib.models.torch.continuous.**ActorProb**(*preprocess_net: Module*, *action_shape: Sequence[int]*, *hidden_sizes: Sequence[int] = ()*, *max_action: float = 1.0*, *device: Union[str, int, device] = 'cpu'*, *unbounded: bool = False*, *conditioned_sigma: bool = False*, *preprocess_net_output_dim: Optional[int] = None*)

Bases: Module

Simple actor network (output with a Gauss distribution). :param preprocess_net: a self-defined preprocess_net which output a

flattened hidden state.

Parameters

- **action_shape** – a sequence of int for the shape of action.
- **hidden_sizes** – a sequence of int for constructing the MLP after preprocess_net. Default to empty sequence (where the MLP now contains only a single linear layer).
- **max_action** (*float*) – the scale for the final action logits. Default to 1.
- **unbounded** (*bool*) – whether to apply tanh activation on final logits. Default to False.
- **conditioned_sigma** (*bool*) – True when sigma is calculated from the input, False when sigma is an independent parameter. Default to False.
- **preprocess_net_output_dim** (*int*) – the output dimension of preprocess_net.

For advanced usage (how to customize the network), please refer to build_the_network. .. seealso:

```
Please refer to :class:`~tianshou.utils.net.common.Net` as an instance
of how preprocess_net is suggested to be defined.
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, *state: Optional[Any] = None*, *info: Dict[str, Any] = {}*) → Tuple[Tuple[Tensor, Tensor], Any]

Mapping: obs -> logits -> (mu, sigma).

training: bool

class malib.models.torch.continuous.**Critic**(*preprocess_net: Module*, *hidden_sizes: Sequence[int] = ()*, *device: Union[str, int, device] = 'cpu'*, *preprocess_net_output_dim: Optional[int] = None*)

Bases: Module

Simple critic network. Will create an actor operated in continuous action space with structure of preprocess_net —> 1(q value). :param preprocess_net: a self-defined preprocess_net which output a

flattened hidden state.

Parameters

- **hidden_sizes** – a sequence of int for constructing the MLP after preprocess_net. Default to empty sequence (where the MLP now contains only a single linear layer).
- **preprocess_net_output_dim** (*int*) – the output dimension of preprocess_net.

For advanced usage (how to customize the network), please refer to build_the_network. .. seealso:

```
Please refer to :class:`~tianshou.utils.net.common.Net` as an instance
of how preprocess_net is suggested to be defined.
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, *act: Optional[Union[ndarray, Tensor]] = None*, *info: Dict[str, Any] = {}*) → Tensor

Mapping: (s, a) -> logits -> Q(s, a).

training: bool

class malib.models.torch.continuous.**Perturbation**(*preprocess_net: Module*, *max_action: float*, *device: Union[str, int, device] = 'cpu'*, *phi: float = 0.05*)

Bases: Module

Implementation of perturbation network in BCQ algorithm. Given a state and action, it can generate perturbed action. :param torch.nn.Module preprocess_net: a self-defined preprocess_net which output a

flattened hidden state.

Parameters

- **max_action** (*float*) – the maximum value of each dimension of action.
- **device** (*Union[str, int, torch.device]*) – which device to create this model on. Default to cpu.
- **phi** (*float*) – max perturbation parameter for BCQ. Default to 0.05.

For advanced usage (how to customize the network), please refer to build_the_network. .. seealso:

```
You can refer to `examples/offline/offline_bcq.py` to see how to use it.
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*state: Tensor*, *action: Tensor*) → Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class malib.models.torch.continuous.**RecurrentActorProb**(*layer_num: int*, *state_shape: Sequence[int]*, *action_shape: Sequence[int]*, *hidden_layer_size: int = 128*, *max_action: float = 1.0*, *device: Union[str, int, device] = 'cpu'*, *unbounded: bool = False*, *conditioned_sigma: bool = False*)

Bases: Module

Recurrent version of ActorProb. For advanced usage (how to customize the network), please refer to build_the_network.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, *state: Optional[Dict[str, Tensor]] = None*, *info: Dict[str, Any] = {}*) → Tuple[Tuple[Tensor, Tensor], Dict[str, Tensor]]

Almost the same as `Recurrent`.

training: bool

class malib.models.torch.continuous.**RecurrentCritic**(*layer_num: int*, *state_shape: Sequence[int]*, *action_shape: Sequence[int] = [0]*, *device: Union[str, int, device] = 'cpu'*, *hidden_layer_size: int = 128*)

Bases: `Module`

Recurrent version of Critic. For advanced usage (how to customize the network), please refer to build_the_network.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, *act: Optional[Union[ndarray, Tensor]] = None*, *info: Dict[str, Any] = {}*) → Tensor

Almost the same as `Recurrent`.

training: bool

class malib.models.torch.continuous.**VAE**(*encoder: Module*, *decoder: Module*, *hidden_dim: int*, *latent_dim: int*, *max_action: float*, *device: Union[str, device] = 'cpu'*)

Bases: `Module`

Implementation of VAE. It models the distribution of action. Given a state, it can generate actions similar to those in batch. It is used in BCQ algorithm. :param torch.nn.Module encoder: the encoder in VAE. Its input_dim must be

state_dim + action_dim, and output_dim must be hidden_dim.

Parameters

- **decoder** (*torch.nn.Module*) – the decoder in VAE. Its input_dim must be state_dim + latent_dim, and output_dim must be action_dim.
- **hidden_dim** (*int*) – the size of the last linear-layer in encoder.
- **latent_dim** (*int*) – the size of latent layer.
- **max_action** (*float*) – the maximum value of each dimension of action.
- **device** (*Union[str, torch.device]*) – which device to create this model on. Default to "cpu".

For advanced usage (how to customize the network), please refer to build_the_network. .. seealso:

```
You can refer to `examples/offline/offline_bcq.py` to see how to use it.
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

decode(*state: Tensor*, *latent_z: Optional[Tensor] = None*) → Tensor

forward(*state: Tensor*, *action: Tensor*) → Tuple[Tensor, Tensor, Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: bool

malib.models.torch.discrete module

class malib.models.torch.discrete.**Actor**(*preprocess_net: Module*, *action_shape: Sequence[int]*, *hidden_sizes: Sequence[int] = ()*, *softmax_output: bool = True*, *preprocess_net_output_dim: Optional[int] = None*, *device: Union[str, int, device] = 'cpu'*)

Bases: `Module`

Simple actor network. Will create an actor operated in discrete action space with structure of preprocess_net —> action_shape. :param preprocess_net: a self-defined preprocess_net which output a

flattened hidden state.

Parameters

- **action_shape** – a sequence of int for the shape of action.
- **hidden_sizes** – a sequence of int for constructing the MLP after preprocess_net. Default to empty sequence (where the MLP now contains only a single linear layer).
- **softmax_output** (*bool*) – whether to apply a softmax layer over the last layer's output.
- **preprocess_net_output_dim** (*int*) – the output dimension of preprocess_net.

For advanced usage (how to customize the network), please refer to build_the_network. .. seealso:

```
Please refer to :class:`~tianshou.utils.net.common.Net` as an instance
of how preprocess_net is suggested to be defined.
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, *state: Optional[Any] = None*, *info: Dict[str, Any] = {}*) → Tuple[Tensor, Any]

Mapping: s -> Q(s, *).

training: bool

class malib.models.torch.discrete.**CosineEmbeddingNetwork**(*num_cosines: int*, *embedding_dim: int*)

Bases: `Module`

Cosine embedding network for IQN. Convert a scalar in [0, 1] to a list of n-dim vectors. :param num_cosines: the number of cosines used for the embedding. :param embedding_dim: the dimension of the embedding/output. .. note:

```
From https://github.com/ku2482/fqf-iqn-qrdqn.pytorch/blob/master
/fqf_iqn_qrdqn/network.py .
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*taus: Tensor*) → Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class malib.models.torch.discrete.**Critic**(*preprocess_net: Module*, *hidden_sizes: Sequence[int] = ()*, *last_size: int = 1*, *preprocess_net_output_dim: Optional[int] = None*, *device: Union[str, int, device] = 'cpu'*)

Bases: `Module`

Simple critic network. Will create an actor operated in discrete action space with structure of preprocess_net —> 1(q value). :param preprocess_net: a self-defined preprocess_net which output a

flattened hidden state.

Parameters

- **hidden_sizes** – a sequence of int for constructing the MLP after preprocess_net. Default to empty sequence (where the MLP now contains only a single linear layer).
- **last_size** (*int*) – the output dimension of Critic network. Default to 1.
- **preprocess_net_output_dim** (*int*) – the output dimension of preprocess_net.

For advanced usage (how to customize the network), please refer to build_the_network. .. seealso:

```
Please refer to :class:`~tianshou.utils.net.common.Net` as an instance
of how preprocess_net is suggested to be defined.
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, ***kwargs: Any*) → Tensor

Mapping: s -> V(s).

training: bool

class malib.models.torch.discrete.**FractionProposalNetwork**(*num_fractions: int*, *embedding_dim: int*)

Bases: `Module`

Fraction proposal network for FQF. :param num_fractions: the number of factions to propose. :param embedding_dim: the dimension of the embedding/input. .. note:

```
Adapted from https://github.com/ku2482/fqf-iqn-qrdqn.pytorch/blob/master
/fqf_iqn_qrdqn/network.py .
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs_embeddings: Tensor*) → Tuple[Tensor, Tensor, Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class malib.models.torch.discrete.**ImplicitQuantileNetwork**(*preprocess_net: Module*, *action_shape: Sequence[int]*, *hidden_sizes: Sequence[int] = ()*, *num_cosines: int = 64*, *preprocess_net_output_dim: Optional[int] = None*, *device: Union[str, int, device] = 'cpu'*)

Bases: *Critic*

Implicit Quantile Network. :param preprocess_net: a self-defined preprocess_net which output a

flattened hidden state.

Parameters

- **action_dim** (*int*) – the dimension of action space.
- **hidden_sizes** – a sequence of int for constructing the MLP after preprocess_net. Default to empty sequence (where the MLP now contains only a single linear layer).
- **num_cosines** (*int*) – the number of cosines to use for cosine embedding. Default to 64.
- **preprocess_net_output_dim** (*int*) – the output dimension of preprocess_net.

Note: Although this class inherits Critic, it is actually a quantile Q-Network with output shape (batch_size, action_dim, sample_size). The second item of the first return value is tau vector.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, *sample_size: int*, ***kwargs: Any*) → Tuple[Any, Tensor]

Mapping: s -> Q(s, *).

training: bool

class malib.models.torch.discrete.**IntrinsicCuriosityModule**(*feature_net: Module*, *feature_dim: int*, *action_dim: int*, *hidden_sizes: Sequence[int] = ()*, *device: Union[str, device] = 'cpu'*)

Bases: `Module`

Implementation of Intrinsic Curiosity Module. arXiv:1705.05363. :param torch.nn.Module feature_net: a self-defined feature_net which output a

flattened hidden state.

Parameters

- **feature_dim** (*int*) – input dimension of the feature net.
- **action_dim** (*int*) – dimension of the action space.
- **hidden_sizes** – hidden layer sizes for forward and inverse models.
- **device** – device for the module.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*s1: Union[ndarray, Tensor]*, *act: Union[ndarray, Tensor]*, *s2: Union[ndarray, Tensor]*, ***kwargs: Any*) → Tuple[Tensor, Tensor]

Mapping: s1, act, s2 -> mse_loss, act_hat.

training: bool

class malib.models.torch.discrete.**NoisyLinear**(*in_features: int*, *out_features: int*, *noisy_std: float = 0.5*)

Bases: Module

Implementation of Noisy Networks. arXiv:1706.10295. :param int in_features: the number of input features. :param int out_features: the number of output features. :param float noisy_std: initial standard deviation of noisy linear layers. .. note:

```
Adapted from https://github.com/ku2482/fqf-iqn-qrdqn.pytorch/blob/master
/fqf_iqn_qrdqn/network.py .
```

Initializes internal Module state, shared by both nn.Module and ScriptModule.

f(*x: Tensor*) → Tensor

forward(*x: Tensor*) → Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

reset() → None

sample() → None

training: bool

malib.models.torch.discrete.**sample_noise**(*model: Module*) → bool

Sample the random noises of NoisyLinear modules in the model. :param model: a PyTorch module which may have NoisyLinear submodules. :returns: True if model has at least one NoisyLinear submodule;

otherwise, False.

malib.models.torch.net module

class malib.models.torch.net.**ActorCritic**(*actor: Module*, *critic: Module*)

Bases: Module

An actor-critic network for parsing parameters. Using actor_critic.parameters() instead of set.union or list+list to avoid issue #449. :param nn.Module actor: the actor network. :param nn.Module critic: the critic network.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

class malib.models.torch.net.**DataParallelNet**(*net: Module*)

Bases: Module

DataParallel wrapper for training agent with multi-GPU. This class does only the conversion of input data type, from numpy array to torch's Tensor. If the input is a nested dictionary, the user should create a similar class to do the same thing. :param nn.Module net: the network to be distributed in different GPUs.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, **args: Any*, ***kwargs: Any*) → Tuple[Any, Any]

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class malib.models.torch.net.**MLP**(*input_dim: int, output_dim: int = 0, hidden_sizes: ~typing.Sequence[int] = (), norm_layer: ~typing.Optional[~typing.Union[~typing.Type[~torch.nn.modules.module.Module], ~typing.Sequence[~typing.Type[~torch.nn.modules.module.Module]]]] = None, activation: ~typing.Optional[~typing.Union[~typing.Type[~torch.nn.modules.module.Module], ~typing.Sequence[~typing.Type[~torch.nn.modules.module.Module]]]] = <class 'torch.nn.modules.activation.ReLU'>, device: ~typing.Optional[~typing.Union[str, int, ~torch.device]] = None, linear_layer: ~typing.Type[~torch.nn.modules.linear.Linear] = <class 'torch.nn.modules.linear.Linear'>*)

Bases: Module

Create a MLP.

Parameters

- **input_dim** (*int*) – dimension of the input vector.
- **output_dim** (*int, optional*) – dimension of the output vector. If set to 0, there is no final linear layer. Defaults to 0.
- **hidden_sizes** (*Sequence[int], optional*) – shape of MLP passed in as a list, not including input_dim and output_dim. Defaults to ().

- **norm_layer** (*Optional[Union[ModuleType, Sequence[ModuleType]]], optional*) – use which normalization before activation, e.g., nn.LayerNorm and nn.BatchNorm1d. Default to no normalization. You can also pass a list of normalization modules with the same length of hidden_sizes, to use different normalization module in different layers. Default to no normalization. Defaults to None.
- **activation** (*Optional[Union[ModuleType, Sequence[ModuleType]]], optional*) – which activation to use after each layer, can be both the same activation for all layers if passed in nn.Module, or different activation for different Modules if passed in a list. Defaults to nn.ReLU.
- **device** (*Optional[Union[str, int, torch.device]], optional*) – which device to create this model on. Defaults to None.
- **linear_layer** (*Type[nn.Linear], optional*) – use this module as linear layer. Defaults to nn.Linear.

forward(*obs: Union[ndarray, Tensor]*) → Tensor

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class malib.models.torch.net.**Net**(*state_shape: ~typing.Union[int, ~typing.Sequence[int]], action_shape: ~typing.Union[int, ~typing.Sequence[int]] = 0, hidden_sizes: ~typing.Sequence[int] = (), norm_layer: ~typing.Optional[~typing.Type[~torch.nn.modules.module.Module]] = None, activation: ~typing.Optional[~typing.Type[~torch.nn.modules.module.Module]] = <class 'torch.nn.modules.activation.ReLU'>, device: ~typing.Union[str, int, ~torch.device] = 'cpu', softmax: bool = False, concat: bool = False, num_atoms: int = 1, dueling_param: ~typing.Optional[~typing.Tuple[~typing.Dict[str, ~typing.Any], ~typing.Dict[str, ~typing.Any]]] = None*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, *state: Optional[Any] = None*, *info: Dict[str, Any] = {}*) → Tuple[Tensor, Any]

Mapping: obs -> flatten (inside MLP)-> logits.

training: bool

class malib.models.torch.net.**Recurrent**(*layer_num: int*, *state_shape: Union[int, Sequence[int]]*, *action_shape: Union[int, Sequence[int]]*, *device: Union[str, int, device] = 'cpu'*, *hidden_layer_size: int = 128*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*obs: Union[ndarray, Tensor]*, *state: Optional[Dict[str, Tensor]] = None*, *info: Dict[str, Any] = {}*) → Tuple[Tensor, Dict[str, Tensor]]

Mapping: obs -> flatten -> logits. In the evaluation mode, *obs* should be with shape `[bsz, dim]`; in the training mode, *obs* should be with shape `[bsz, len, dim]`. See the code and comment for more detail.

training: bool

malib.models.torch.net.**make_net**(*observation_space: Space*, *action_space: Space*, *device: Type[device]*, *net_type: Optional[str] = None*, ***kwargs*) → Module

Create a network instance with specific network configuration.

Parameters

- **observation_space** (*gym.Space*) – The observation space used to determine which network type will be used, if net_type is not be specified
- **action_space** (*gym.Space*) – The action space will be used to determine the network output dim, if *output_dim* or *action_shape* is not given in kwargs
- **device** (*Device*) – Indicates device allocated.
- **net_type** (*str, optional*) – Indicates the network type, could be one from {mlp, net, rnn, actor_critic, data_parallel}

Raises

ValueError – Unexpected network type.

Returns

A network instance.

Return type

nn.Module

malib.models.torch.net.**miniblock**(*input_size: int*, *output_size: int = 0*, *norm_layer: ~typing.Optional[~typing.Type[~torch.nn.modules.module.Module]] = None*, *activation: ~typing.Optional[~typing.Type[~torch.nn.modules.module.Module]] = None*, *linear_layer: ~typing.Type[~torch.nn.modules.linear.Linear] = <class 'torch.nn.modules.linear.Linear'>*) → List[Module]

Construct a miniblock with given input/output-size, norm layer and activation.

Parameters

- **input_size** (*int*) – The input size.
- **output_size** (*int, optional*) – The output size. Defaults to 0.
- **norm_layer** (*Optional[ModuleType], optional*) – A nn.Module as normal layer. Defaults to None.
- **activation** (*Optional[ModuleType], optional*) – A nn.Module as active layer. Defaults to None.
- **linear_layer** (*Type[nn.Linear], optional*) – A nn.Module as linear layer. Defaults to nn.Linear.

Returns

A list of layers.

Return type

List[nn.Module]

CHAPTER

ELEVEN

MALIB.REMOTE PACKAGE

11.1 Submodules

11.2 malib.remote.interface module

class malib.remote.interface.**RemoteInterface**

Bases: object

classmethod as_remote(*num_cpus: Optional[int] = None*, *num_gpus: Optional[int] = None*, *memory: Optional[int] = None*, *object_store_memory: Optional[int] = None*, *resources: Optional[dict] = None*) → type

Return a remote class for Actor initialization

is_running()

set_running(*value*)

stop_pending_tasks()

External object can call this method to stop all pending tasks.

CHAPTER

TWELVE

MALIB.RL PACKAGE

12.1 Subpackages

12.1.1 malib.rl.a2c package

Submodules

malib.rl.a2c.config module

malib.rl.a2c.policy module

class malib.rl.a2c.policy.**A2CPolicy**(*observation_space: Space*, *action_space: Space*, *model_config: Dict[str, Any]*, *custom_config: Dict[str, Any]*, ***kwargs*)

Bases: *PGPolicy*

Build a REINFORCE policy whose input and output dims are determined by observation_space and action_space, respectively.

Parameters

- **observation_space** (*spaces.Space*) – The observation space.
- **action_space** (*spaces.Space*) – The action space.
- **model_config** (*Dict[str, Any]*) – The model configuration dict.
- **custom_config** (*Dict[str, Any]*) – The custom configuration dict.
- **is_fixed** (*bool, optional*) – Indicates fixed policy or trainable policy. Defaults to False.

Raises

- **NotImplementedError** – Does not support other action space type settings except Box and Discrete.
- **TypeError** – Unexpected action space.

value_function(*observation: Tensor*, *evaluate: bool*, ***kwargs*)

Compute values of critic.

malib.rl.a2c.trainer module

class malib.rl.a2c.trainer.**A2CTrainer**(*training_config: Dict[str, Any]*, *policy_instance: Optional[Policy] = None*)

Bases: *Trainer*

Initialize a trainer for a type of policies.

Parameters

- **learning_mode** (*str*) – Learning mode inidication, could be *off_policy* or *on_policy*.
- **training_config** (*Dict[str, Any], optional*) – The training configuration. Defaults to None.
- **policy_instance** (*Policy, optional*) – A policy instance, if None, we must reset it. Defaults to None.

post_process(*batch:* Batch, *agent_filter: Sequence[str]*) → *Batch*

Batch post processing here.

Parameters
batch (Batch) – Sampled batch.

Raises
NotImplementedError – Not implemented.

Returns
A batch instance.

Return type
Batch

setup()

Set up optimizers here.

train(*batch:* Batch) → Dict[str, float]

Run training, and return info dict.

Parameters
batch (*Union[Dict[AgentID, Batch], Batch]*) – A dict of batch or batch

Returns
A training batch of data.

Return type
Batch

12.1.2 malib.rl.coma package

Submodules

malib.rl.coma.critic module

class malib.rl.coma.critic.**COMADiscreteCritic**(*centralized_obs_space: Space*, *action_space: Space*, *net_type: Optional[str] = None*, *device: str = 'cpu'*, ***kwargs*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*inputs: Union[Dict[str, Batch], Tensor]*) → Union[Tuple[Tensor, Any], Tensor]

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

malib.rl.coma.trainer module

class malib.rl.coma.trainer.**COMATrainer**(*training_config: Dict[str, Any]*, *critic_creator: Callable*, *policy_instance: Optional[Policy] = None*)

Bases: *Trainer*

Initialize a trainer for a type of policies.

Parameters

- **learning_mode** (*str*) – Learning mode inidication, could be *off_policy* or *on_policy*.
- **training_config** (*Dict[str, Any], optional*) – The training configuration. Defaults to None.
- **policy_instance** (*Policy, optional*) – A policy instance, if None, we must reset it. Defaults to None.

create_joint_action(*n_agents*, *batch_size*, *time_step*, *actions*)

post_process(*batch: Dict[str, Batch]*, *agent_filter: Sequence[str]*) → *Batch*

Stack batches in agent wise.

Parameters

- **batch** (*Dict[str, Any]*) – A dict of agent batches.
- **agent_filter** (*Sequence[AgentID]*) – A list of agent filter.

Returns
Batch

Return type
Batch

setup()

Set up optimizers here.

train(*batch: Batch*) → Dict[str, float]

Run training, and return info dict.

Parameters
batch (*Union[Dict[AgentID, Batch], Batch]*) – A dict of batch or batch

Returns
A training batch of data.

Return type
Batch

train_critic(*batch:* Batch)

12.1.3 malib.rl.common package

Submodules

malib.rl.common.misc module

malib.rl.common.misc.**gumbel_softmax**(*logits: Tensor*, *temperature=1.0*, *mask: Optional[Tensor] = None*, *explore=False*) → Tensor

Convert a softmax to one hot but gradients computation will be kept.

Parameters

- **logits** (*torch.Tensor*) – Raw logits tensor.
- **temperature** (*float, optional*) – Temperature to control the distribution density. Defaults to 1.0.
- **mask** (*torch.Tensor, optional*) – Action masking. Defaults to None.
- **explore** (*bool, optional*) – Enable noise adding or not. Defaults to True.

Returns
Genearted gumbel softmax, shaped as (batch_size, n_classes)

Return type
torch.Tensor

malib.rl.common.misc.**masked_logits**(*logits: Tensor*, *mask: Tensor*)

malib.rl.common.misc.**onehot_from_logits**(*logits: Tensor*, *eps=0.0*)

Given batch of logits, return one-hot sample using epsilon greedy strategy (based on given epsilon)

malib.rl.common.misc.**sample_gumbel**(*shape: ~torch.Size*, *eps: float = 1e-20*, *tens_type: ~typing.Type = <class 'torch.FloatTensor'>*) → Tensor

Sample noise from an uniform distribution withe a given shape. Note the returned tensor is deactivated for gradients computation.

Parameters

- **shape** (*torch.Size*) – Target shape.
- **eps** (*float, optional*) – Tolerance to avoid NaN. Defaults to 1e-20.
- **tens_type** (*Type, optional*) – Indicates the data type of the sampled noise. Defaults to torch.FloatTensor.

Returns
A tensor as sampled noise.

Return type
torch.Tensor

malib.rl.common.misc.**soft_update**(*target: Module*, *source: Module*, *tau: float*)

Perform soft update.

Parameters

- **target** (*torch.nn.Module*) – Net to copy parameters to

- **source** (*torch.nn.Module*) – Net whose parameters to copy
- **tau** (*float*) – Range form 0 to 1, weight factor for update

malib.rl.common.misc.**softmax**(*logits: Tensor*, *temperature: float*, *mask: Optional[Tensor] = None*, *explore: bool = True*) → Tensor

Apply softmax to the given logits. With distribution density control and optional exploration noise.

Parameters

- **logits** (*torch.Tensor*) – Logits tensor.
- **temperature** (*float*) – Temperature controls the distribution density.
- **mask** (*torch.Tensor, optional*) – Applying action mask if not None. Defaults to None.
- **explore** (*bool, optional*) – Add noise to the generated distribution or not. Defaults to True.

Raises

TypeError – Logits should be a *torch.Tensor*.

Returns

softmax tensor, shaped as (batch_size, n_classes).

Return type

torch.Tensor

malib.rl.common.policy module

class malib.rl.common.policy.**Policy**(*observation_space*, *action_space*, *model_config*, *custom_config*, ***kwargs*)

Bases: object

property actor

abstract compute_action(*observation: Tensor*, *act_mask: Optional[Tensor]*, *evaluate: bool*, *hidden_state: Optional[Any] = None*, ***kwargs*) → Tuple[Any, Any, Any, Any]

coordinate(*state: Dict[str, Tensor]*, *message: Any*) → Any

Coordinate with other agents here

classmethod copy(*instance*, *replacement: Dict*)

property critic

property custom_config: Dict[str, Any]

deregister_state(*name: str*)

property device: str

get_initial_state(*batch_size: Optional[int] = None*)

load(*path: str*)

load_state_dict(*state_dict: Dict[str, Any]*)

Load state dict outside.

Parameters

state_dict (*Dict[str, Any]*) – A dict of states.

property model_config

parameters() → Dict[str, Dict]

Return trainable parameters.

property preprocessor

register_state(*obj: Any*, *name: str*) → None

Register state of obj. Called in init function to register model states.

Example

```
>>> class CustomPolicy(Policy):
...     def __init__(
...         self,
...         registered_name,
...         observation_space,
...         action_space,
...         model_config,
...         custom_config
...     ):
...     # ...
...     actor = MLP(...)
...     self.register_state(actor, "actor")
```

Parameters

- **obj** (*Any*) – Any object, for non *torch.nn.Module*, it will be wrapped as a *Simpleobject*.
- **name** (*str*) – Humanreadable name, to identify states.

Raises

errors.RepeatedAssignError – [description]

property registered_networks: Dict[str, Module]

reset(***kwargs*)

Reset parameters or behavior policies.

save(*path*, *global_step=0*, *hard: bool = False*)

state_dict(*device=None*)

Return state dict in real time

property target_actor

property target_critic

to(*device: Optional[str] = None*, *use_copy: bool = False*) → *Policy*

Convert policy to a given device. If *use_copy*, then return a copy. If device is None, do not change device.

Parameters

- **device** (*str*) – Device identifier.
- **use_copy** (*bool, optional*) – User copy or not. Defaults to False.

Raises

NotImplementedError – Not implemented error.

Returns

A policy instance

Return type

Policy

update_parameters(*parameter_dict: Dict[str, Any]*)

Update local parameters with given parameter dict.

Parameters

parameter_dict (*Dict[str, Parameter]*) – A dict of paramters

class malib.rl.common.policy.**SimpleObject**(*obj*, *name*)

Bases: object

load_state_dict(*v*)

state_dict()

malib.rl.common.trainer module

class malib.rl.common.trainer.**Trainer**(*training_config: Dict[str, Any]*, *policy_instance: Optional[Policy] = None*)

Bases: object

Initialize a trainer for a type of policies.

Parameters

- **learning_mode** (*str*) – Learning mode inidication, could be *off_policy* or *on_policy*.
- **training_config** (*Dict[str, Any], optional*) – The training configuration. Defaults to None.
- **policy_instance** (*Policy, optional*) – A policy instance, if None, we must reset it. Defaults to None.

property counter

parameters()

property policy

post_process(*batch: Batch*, *agent_filter: Sequence[str]*) → *Batch*

Batch post processing here.

Parameters

batch (*Batch*) – Sampled batch.

Raises
NotImplementedError – Not implemented.

Returns
A batch instance.

Return type
Batch

reset(*policy_instance=None*, *configs=None*, *learning_mode: Optional[str] = None*)

Reset current trainer, with given policy instance, training configuration or learning mode.

Note: Becareful to reset the learning mode, since it will change the sample behavior. Specifically, the *on_policy* mode will sample datas sequntially, which will return a *torch.DataLoader* to the method *self.train*. For the *off_policy* case, the sampler will sample data randomly, which will return a *dict* to

---

Parameters

- **policy_instance** (*Policy, optional*) – A policy instance. Defaults to None.
- **configs** (*Dict[str, Any], optional*) – A training configuration used to update existing one. Defaults to None.
- **learning_mode** (*str, optional*) – Learning mode, could be *off_policy* or *on_policy*. Defaults to None.

abstract setup()

Set up optimizers here.

step_counter()

abstract train(*batch:* Batch) → Dict[str, float]

Run training, and return info dict.

Parameters
batch (*Union[Dict[AgentID, Batch], Batch]*) – A dict of batch or batch

Returns
A training batch of data.

Return type
Batch

property training_config: Dict[str, Any]

12.1.4 malib.rl.discrete_sac package

Submodules

malib.rl.discrete_sac.policy module

malib.rl.discrete_sac.trainer module

12.1.5 malib.rl.dqn package

Submodules

malib.rl.dqn.config module

malib.rl.dqn.policy module

class malib.rl.dqn.policy.**DQNPolicy**(*observation_space: Space*, *action_space: Space*, *model_config: Dict[str, Any]*, *custom_config: Dict[str, Any]*, ***kwargs*)

Bases: *Policy*

compute_action(*observation: Tensor*, *act_mask: Optional[Tensor]*, *evaluate: bool*, *hidden_state: Optional[Any] = None*, ***kwargs*)

Compute action in rollout stage. Do not support vector mode yet.

Parameters

- **observation** (*DataArray*) – The observation batched data with shape=(n_batch, obs_shape).
- **act_mask** (*DataArray*) – The action mask batched with shape=(n_batch, mask_shape).
- **evaluate** (*bool*) – Turn off exploration or not.
- **state** (*Any, Optional*) – The hidden state. Default by None.

property eps: float

load(*path: str*)

parameters()

Return trainable parameters.

reset(***kwargs*)

Reset parameters or behavior policies.

save(*path*, *global_step=0*, *hard: bool = False*)

value_function(*observation: Tensor*, *evaluate: bool*, ***kwargs*) → ndarray

malib.rl.dqn.trainer module

class malib.rl.dqn.trainer.**DQNTrainer**(*training_config: Dict[str, Any]*, *policy_instance: Optional[*Policy*] = None*)

Bases: *Trainer*

Initialize a trainer for a type of policies.

Parameters

- **learning_mode** (*str*) – Learning mode inidication, could be *off_policy* or *on_policy*.
- **training_config** (*Dict[str, Any], optional*) – The training configuration. Defaults to None.
- **policy_instance** (*Policy, optional*) – A policy instance, if None, we must reset it. Defaults to None.

post_process(*batch:* Batch, *agent_filter: Sequence[str]*) → *Batch*

Batch post processing here.

Parameters
batch (Batch) – Sampled batch.

Raises
NotImplementedError – Not implemented.

Returns
A batch instance.

Return type
Batch

setup()

Set up optimizers here.

train(*batch:* Batch)

Run training, and return info dict.

Parameters
batch (*Union[Dict[AgentID, Batch], Batch]*) – A dict of batch or batch

Returns
A training batch of data.

Return type
Batch

12.1.6 malib.rl.maddpg package

Submodules

malib.rl.maddpg.loss module

malib.rl.maddpg.trainer module

12.1.7 malib.rl.mappo package

Submodules

malib.rl.mappo.config module

malib.rl.mappo.policy module

malib.rl.mappo.trainer module

12.1.8 malib.rl.pg package

Submodules

malib.rl.pg.config module

malib.rl.pg.policy module

class malib.rl.pg.policy.**PGPolicy**(*observation_space: Space*, *action_space: Space*, *model_config: Dict[str, Any]*, *custom_config: Dict[str, Any]*, ***kwargs*)

Bases: *Policy*

Build a REINFORCE policy whose input and output dims are determined by observation_space and action_space, respectively.

Parameters

- **observation_space** (*spaces.Space*) – The observation space.
- **action_space** (*spaces.Space*) – The action space.
- **model_config** (*Dict[str, Any]*) – The model configuration dict.
- **custom_config** (*Dict[str, Any]*) – The custom configuration dict.
- **is_fixed** (*bool, optional*) – Indicates fixed policy or trainable policy. Defaults to False.

Raises

- **NotImplementedError** – Does not support other action space type settings except Box and Discrete.
- **TypeError** – Unexpected action space.

compute_action(*observation: Tensor*, *act_mask: Optional[Tensor]*, *evaluate: bool*, *hidden_state: Optional[Any] = None*, ***kwargs*) → Tuple[Any, Any, Any, Any]

value_function(*observation: Tensor*, *evaluate: bool*, ***kwargs*)

Compute values of critic.

malib.rl.pg.trainer module

class malib.rl.pg.trainer.**PGTrainer**(*training_config: Dict[str, Any]*, *policy_instance: Optional[Policy] = None*)

Bases: *Trainer*

Initialize a trainer for a type of policies.

Parameters

- **learning_mode** (*str*) – Learning mode inidication, could be *off_policy* or *on_policy*.
- **training_config** (*Dict[str, Any], optional*) – The training configuration. Defaults to None.
- **policy_instance** (*Policy, optional*) – A policy instance, if None, we must reset it. Defaults to None.

post_process(*batch: Batch*, *agent_filter: Sequence[str]*) → *Batch*

Batch post processing here.

Parameters
batch (*Batch*) – Sampled batch.

Raises
NotImplementedError – Not implemented.

Returns
A batch instance.

Return type
Batch

`setup()`

Set up optimizers here.

`train`(*batch:* Batch) → Dict[str, Any]

Run training, and return info dict.

Parameters
batch (*Union[Dict[AgentID, Batch], Batch]*) – A dict of batch or batch

Returns
A training batch of data.

Return type
Batch

12.1.9 malib.rl.ppo package

Submodules

malib.rl.ppo.policy module

malib.rl.ppo.trainer module

12.1.10 malib.rl.qmix package

Submodules

malib.rl.qmix.q_mixer module

malib.rl.qmix.trainer module

12.1.11 malib.rl.random package

Submodules

malib.rl.random.config module

malib.rl.random.policy module

`class` malib.rl.random.policy.**RandomPolicy**(*observation_space: Space*, *action_space: Space*, *model_config: Dict[str, Any]*, *custom_config: Dict[str, Any]*, ***kwargs*)

Bases: *PGPolicy*

Build a REINFORCE policy whose input and output dims are determined by observation_space and action_space, respectively.

Parameters

- **observation_space** (*spaces.Space*) – The observation space.
- **action_space** (*spaces.Space*) – The action space.

- **model_config** (*Dict[str, Any]*) – The model configuration dict.
- **custom_config** (*Dict[str, Any]*) – The custom configuration dict.
- **is_fixed** (*bool, optional*) – Indicates fixed policy or trainable policy. Defaults to False.

Raises

- **NotImplementedError** – Does not support other action space type settings except Box and Discrete.
- **TypeError** – Unexpected action space.

malib.rl.random.random_trainer module

class malib.rl.random.random_trainer.**RandomTrainer**(*training_config: Dict[str, Any]*, *policy_instance: Optional[Policy] = None*)

Bases: *PGTrainer*

Initialize a trainer for a type of policies.

Parameters

- **learning_mode** (*str*) – Learning mode inidication, could be *off_policy* or *on_policy*.
- **training_config** (*Dict[str, Any], optional*) – The training configuration. Defaults to None.
- **policy_instance** (*Policy, optional*) – A policy instance, if None, we must reset it. Defaults to None.

12.1.12 malib.rl.sac package

Submodules

malib.rl.sac.policy module

malib.rl.sac.trainer module

CHAPTER

THIRTEEN

MALIB.ROLLOUT PACKAGE

13.1 Subpackages

13.1.1 malib.rollout.envs package

Subpackages

malib.rollout.envs.gr_football package

Submodules

malib.rollout.envs.gr_football.env module

malib.rollout.envs.gr_football.wrappers module

malib.rollout.envs.gym package

malib.rollout.envs.gym.**env_desc_gen**(***config*)

Submodules

malib.rollout.envs.gym.env module

class malib.rollout.envs.gym.env.**GymEnv**(***configs*)

Bases: *Environment*

Single agent gym envrionment

property action_spaces: Dict[str, Space]

A dict of agent action spaces

close()

property observation_spaces: Dict[str, Space]

A dict of agent observation spaces

property possible_agents: List[str]

Return a list of environment agent ids

render(**args*, ***kwargs*)

reset(*max_step: Optional[int] = None*)

Reset environment and the episode info handler here.

time_step(*actions: Dict[str, Any]*) → Tuple[Dict[str, Any], Dict[str, float], Dict[str, bool], Dict[str, Any]]

Environment stepping logic.

Parameters
actions (*Dict[AgentID, Any]*) – Agent action dict.

Raises
NotImplementedError – Not implmeneted error

Returns
A 4-tuples, listed as (observations, rewards, dones, infos)

Return type
Tuple[Dict[AgentID, Any], Dict[AgentID, float], Dict[AgentID, bool], Dict[AgentID, Any]]

malib.rollout.envs.mdp package

malib.rollout.envs.mdp.**env_desc_gen**(***config*)

Submodules

malib.rollout.envs.mdp.env module

class malib.rollout.envs.mdp.env.**MDPEnvironment**(***configs*)

Bases: *Environment*

property action_spaces: Dict[str, Space]

A dict of agent action spaces

close()

property observation_spaces: Dict[str, Space]

A dict of agent observation spaces

property possible_agents: List[str]

Return a list of environment agent ids

render(**args*, ***kwargs*)

reset(*max_step: Optional[int] = None*) → Union[None, Sequence[Dict[str, Any]]]

Reset environment and the episode info handler here.

seed(*seed: Optional[int] = None*)

time_step(*actions: Dict[str, Any]*) → Tuple[Dict[str, Any], Dict[str, float], Dict[str, bool], Dict[str, Any]]

Environment stepping logic.

Parameters
actions (*Dict[AgentID, Any]*) – Agent action dict.

Raises
NotImplementedError – Not implmeneted error

Returns
A 4-tuples, listed as (observations, rewards, dones, infos)

Return type
Tuple[Dict[AgentID, Any], Dict[AgentID, float], Dict[AgentID, bool], Dict[AgentID, Any]]

malib.rollout.envs.open_spiel package

malib.rollout.envs.open_spiel.**env_desc_gen**(***config*)

Submodules

malib.rollout.envs.open_spiel.env module

malib.rollout.envs.open_spiel.env.**ActionSpace**(*action_spec: Dict*) → Space

Analyzes accepted action spec and returns a truncated action space. :param action_spec: The raw action spec in dict. :type action_spec: types.Dict

Returns
The trucated action space.

Return type
gym.Space

malib.rollout.envs.open_spiel.env.**ObservationSpace**(*observation_spec: Dict*, ***kwargs*) → Dict

Analyzes accepted observation spec and returns a truncated observation space. :param observation_spec: The raw obsevation spec in dict. :type observation_spec: Dict

Returns
The truncated observation space in Dict.

Return type
gym.spaces.Dict

class malib.rollout.envs.open_spiel.env.**OpenSpielEnv**(***configs*)

Bases: *Environment*

property action_spaces: Dict[str, Space]

A dict of agent action spaces

close()

property observation_spaces: Dict[str, Space]

A dict of agent observation spaces

property possible_agents: List[str]

Return a list of environment agent ids

reset(*max_step: Optional[int] = None*)

Reset environment and the episode info handler here.

seed(*seed: Optional[int] = None*)

time_step(*actions: Dict[str, Any]*) → Tuple[Dict[str, Any], Dict[str, float], Dict[str, bool], Dict[str, Any]]

Environment stepping logic.

Parameters
actions (*Dict[AgentID, Any]*) – Agent action dict.

Raises
NotImplementedError – Not implmeneted error

Returns
A 4-tuples, listed as (observations, rewards, dones, infos)

Return type
Tuple[Dict[AgentID, Any], Dict[AgentID, float], Dict[AgentID, bool], Dict[AgentID, Any]]

malib.rollout.envs.pettingzoo package

malib.rollout.envs.pettingzoo.**env_desc_gen**(***config*)

Submodules

malib.rollout.envs.pettingzoo.env module

class malib.rollout.envs.pettingzoo.env.**PettingZooEnv**(***configs*)

Bases: *Environment*

property action_spaces: Dict[str, Space]

A dict of agent action spaces

close()

property observation_spaces: Dict[str, Space]

A dict of agent observation spaces

property parallel_simulate: bool

property possible_agents: List[str]

Return a list of environment agent ids

render(**args*, ***kwargs*)

reset(*max_step: Optional[int] = None*) → Union[None, Sequence[Dict[str, Any]]]

Reset environment and the episode info handler here.

seed(*seed: Optional[int] = None*)

time_step(*actions: Dict[str, Any]*) → Tuple[Dict[str, Any], Dict[str, float], Dict[str, bool], Dict[str, Any]]

Environment stepping logic.

Parameters
actions (*Dict[AgentID, Any]*) – Agent action dict.

Raises
NotImplementedError – Not implmeneted error

Returns
A 4-tuples, listed as (observations, rewards, dones, infos)

Return type
Tuple[Dict[AgentID, Any], Dict[AgentID, float], Dict[AgentID, bool], Dict[AgentID, Any]]

malib.rollout.envs.pettingzoo.scenario_configs_ref module

malib.rollout.envs.sc2 package

Submodules

malib.rollout.envs.sc2.env module

Submodules

malib.rollout.envs.env module

class malib.rollout.envs.env.**Environment**(***configs*)

Bases: object

static action_adapter(*policy_outputs: Dict[str, Dict[str, Any]]*, ***kwargs*)
Convert policy action to environment actions. Default by policy action

property action_spaces: Dict[str, Space]
A dict of agent action spaces

close()

collect_info() → Dict[str, Any]

env_done_check(*agent_dones: Dict[str, bool]*) → bool

property observation_spaces: Dict[str, Space]
A dict of agent observation spaces

property possible_agents: List[str]
Return a list of environment agent ids

record_episode_info_step(*state: Any*, *observations: Dict[str, Any]*, *rewards: Dict[str, Any]*, *dones: Dict[str, bool]*, *infos: Any*)
Analyze timestep and record it as episode information.

Parameters

- **state** (*Any*) – Environment state.
- **observations** (*Dict[AgentID, Any]*) – A dict of agent observations
- **rewards** (*Dict[AgentID, Any]*) – A dict of agent rewards.
- **dones** (*Dict[AgentID, bool]*) – A dict of done signals.
- **infos** (*Any*) – Information.

render(**args*, ***kwargs*)

reset(*max_step: Optional[int] = None*) → Union[None, Sequence[Dict[str, Any]]]
Reset environment and the episode info handler here.

seed(*seed: Optional[int] = None*)

step(*actions: Dict[str, Any]*) → Tuple[Dict[str, Any], Dict[str, Any], Dict[str, float], Dict[str, bool], Any]

Return a 5-tuple as (state, observation, reward, done, info). Each item is a dict maps from agent id to entity.

Note: If state return of this environment is not activated, the return state would be None.

Parameters
actions (*Dict[AgentID, Any]*) – A dict of agent actions.

Returns
A tuple follows the order as (state, observation, reward, done, info).

Return type
Tuple[Dict[AgentID, Any], Dict[AgentID, Any], Dict[AgentID, float], Dict[AgentID, bool], Any]

time_step(*actions: Dict[str, Any]*) → Tuple[Dict[str, Any], Dict[str, float], Dict[str, bool], Dict[str, Any]]

Environment stepping logic.

Parameters
actions (*Dict[AgentID, Any]*) – Agent action dict.

Raises
NotImplementedError – Not implmeneted error

Returns
A 4-tuples, listed as (observations, rewards, dones, infos)

Return type
Tuple[Dict[AgentID, Any], Dict[AgentID, float], Dict[AgentID, bool], Dict[AgentID, Any]]

class malib.rollout.envs.env.**GroupWrapper**(*env: Environment*, *aid_to_gid: Dict[str, str]*, *agent_groups: Dict[str, List[str]]*)

Bases: *Wrapper*

Construct a wrapper for a given enviornment instance.

Parameters
env (Environment) – Environment instance.

action_mask_extract(*raw_observations: Dict[str, Any]*)

property action_spaces: Dict[str, Space]

A dict of agent action spaces

property agent_groups: Dict[str, List[str]]

agent_to_group(*agent_id: str*) → str

Mapping agent id to groupd id.

Parameters
agent_id (*AgentID*) – Agent id.

Returns
Group id.

Return type
str

build_state_from_observation(*agent_observation: Dict[str, Any]*) → Dict[str, ndarray]

Build state from raw observation.

Parameters
agent_observation (*Dict[AgentID, Any]*) – A dict of agent observation.

Raises
NotImplementedError – Not implemented error

Returns
A dict of states.

Return type
Dict[str, np.ndarray]

build_state_spaces() → Dict[str, Space]

Call *self.group_to_agents* to build state space here

env_done_check(*agent_dones: Dict[str, bool]*) → bool

property observation_spaces: Dict[str, Space]

A dict of agent observation spaces

property possible_agents: List[str]

Return a list of environment agent ids

record_episode_info_step(*observations*, *rewards*, *dones*, *infos*)

Analyze timestep and record it as episode information.

Parameters

- **state** (*Any*) – Environment state.
- **observations** (*Dict[AgentID, Any]*) – A dict of agent observations
- **rewards** (*Dict[AgentID, Any]*) – A dict of agent rewards.
- **dones** (*Dict[AgentID, bool]*) – A dict of done signals.
- **infos** (*Any*) – Information.

reset(*max_step: Optional[int] = None*) → Union[None, Dict[str, Dict[str, Any]]]

Reset environment and the episode info handler here.

property state_spaces: Dict[str, Space]

Return a dict of group state spaces.

Note: Users must implement the method *build_state_space*.

Returns
A dict of state spaces.

Return type
Dict[str, gym.Space]

time_step(*actions: Dict[str, Any]*)

Environment stepping logic.

Parameters
actions (*Dict[AgentID, Any]*) – Agent action dict.

Raises
NotImplementedError – Not implmeneted error

Returns
A 4-tuples, listed as (observations, rewards, dones, infos)

Return type
Tuple[Dict[AgentID, Any], Dict[AgentID, float], Dict[AgentID, bool], Dict[AgentID, Any]]

class malib.rollout.envs.env.**Wrapper**(*env:* Environment)

Bases: *Environment*

Wraps the environment to allow a modular transformation

Construct a wrapper for a given enviornment instance.

Parameters
env (Environment) – Environment instance.

property **action_spaces: Dict[str, Space]**
A dict of agent action spaces

close()

collect_info() → Dict[str, Any]

property **observation_spaces: Dict[str, Space]**
A dict of agent observation spaces

property **possible_agents: List[str]**
Return a list of environment agent ids

render(**args*, ***kwargs*)

reset(*max_step: Optional[int] = None*) → Union[None, Tuple[Dict[str, Any]]]
Reset environment and the episode info handler here.

seed(*seed: Optional[int] = None*)

step(*actions: Dict[str, Any]*) → Tuple[Dict[str, Any], Dict[str, Any], Dict[str, float], Dict[str, bool], Any]
Return a 5-tuple as (state, observation, reward, done, info). Each item is a dict maps from agent id to entity.

Note: If state return of this environment is not activated, the return state would be None.

Parameters
actions (*Dict[AgentID, Any]*) – A dict of agent actions.

Returns
A tuple follows the order as (state, observation, reward, done, info).

Return type
Tuple[Dict[AgentID, Any], Dict[AgentID, Any], Dict[AgentID, float], Dict[AgentID, bool], Any]

malib.rollout.envs.vector_env module

13.1.2 malib.rollout.inference package

Subpackages

malib.rollout.inference.ray package

Submodules

malib.rollout.inference.ray.client module

malib.rollout.inference.ray.server module

class malib.rollout.inference.ray.server.**ClientHandler**(*sender*, *recver*, *runtime_config*, *rnn_states*)

Bases: tuple

Create new instance of ClientHandler(sender, recver, runtime_config, rnn_states)

property recver

Alias for field number 1

property rnn_states

Alias for field number 3

property runtime_config

Alias for field number 2

property sender

Alias for field number 0

class malib.rollout.inference.ray.server.**RayInferenceWorkerSet**(*agent_id: str*, *observation_space: Space*, *action_space: Space*, *parameter_server: ParameterServer*, *governed_agents: List[str]*)

Bases: *RemoteInterface*

Create ray-based inference server.

Parameters

- **agent_id** (*AgentID*) – Runtime agent id, not environment agent id.
- **observation_space** (*gym.Space*) – Observation space related to the governed environment agents.
- **action_space** (*gym.Space*) – Action space related to the governed environment agents.
- **parameter_server** (ParameterServer) – Parameter server.
- **governed_agents** (*List[AgentID]*) – A list of environment agents.

compute_action(*dataframes: List[*DataFrame*]*, *runtime_config: Dict[str, Any]*) → List[*DataFrame*]

save(*model_dir: str*) → None

shutdown()

Submodules

malib.rollout.inference.utils module

13.2 Submodules

13.3 malib.rollout.manager module

13.4 malib.rollout.pb_rolloutworker module

13.5 malib.rollout.rolloutworker module

CHAPTER

FOURTEEN

MALIB.SCENARIOS PACKAGE

14.1 Submodules

14.2 malib.scenarios.league_training_scenario module

14.3 malib.scenarios.marl_scenario module

14.4 malib.scenarios.psro_scenario module

14.5 malib.scenarios.scenario module

class malib.scenarios.scenario.**Scenario**(*name: str*, *log_dir: str*, *env_desc: Dict[str, Any]*, *algorithms: Dict[str, Any]*, *agent_mapping_func: function*, *training_config: Dict[str, Any]*, *rollout_config: Dict[str, Any]*, *stopping_conditions: Dict[str, Any]*, *dataset_config: Dict[str, Any]*, *parameter_server_config: Dict[str, Any]*)

Bases: ABC

copy()

with_updates(**kwargs*) → *Scenario*

CHAPTER

FIFTEEN

MALIB.UTILS PACKAGE

15.1 Submodules

15.2 malib.utils.data module

class malib.utils.data.**Postprocessor**

Bases: object

static compute_episodic_return(*batch: Dict[str, Any]*, *state_value: Optional[ndarray] = None*, *next_state_value: Optional[ndarray] = None*, *gamma: float = 0.99*, *gae_lambda: float = 0.95*)

static gae_return(*state_value*, *next_state_value*, *reward*, *done*, *gamma: float = 0.99*, *gae_lambda: float = 0.95*)

malib.utils.data.**to_torch**(*x: Any*, *dtype: Optional[dtype] = None*, *device: Union[str, int, device] = 'cpu'*) → Tensor

Return an object without np.ndarray.

15.3 malib.utils.episode module

class malib.utils.episode.**Episode**(*agents: List[str]*, *processors=None*)

Bases: object

Multi-agent episode tracking

ACC_REWARD = 'accumulate_reward'

ACTION = 'act'

ACTION_DIST = 'act_dist'

ACTION_LOGITS = 'act_logits'

ACTION_MASK = 'act_mask'

ADVANTAGE = 'advantage'

CUR_OBS = 'obs'

CUR_STATE = 'state'

DONE = 'done'

INFO = 'infos'

LAST_REWARD = 'last_reward'

NEXT_ACTION_MASK = 'act_mask_next'

NEXT_OBS = 'obs_next'

NEXT_STATE = 'state_next'

PRE_DONE = 'pre_done'

PRE_REWARD = 'pre_rew'

REWARD = 'rew'

RNN_STATE = 'rnn_state'

STATE_ACTION_VALUE = 'state_action_value_estimation'

STATE_VALUE = 'state_value_estimation'

STATE_VALUE_TARGET = 'state_value_target'

record(*data: Dict[str, Dict[str, Any]]*, *agent_first: bool*, *ignore_keys={}*)

Save a transiton. The given transition is a sub sequence of (obs, action_mask, reward, done, info). Users specify ignore keys to filter keys.

Parameters

- **data** (*Dict[str, Dict[AgentID, Any]]*) – A transition.
- **ignore_keys** (*dict, optional*) – . Defaults to {}.

to_numpy() → Dict[str, Dict[str, ndarray]]

Convert episode to numpy array-like data.

class malib.utils.episode.**NewEpisodeDict**

Bases: defaultdict

Episode dict, for trajectory tracking for a bunch of environments.

record(*data: Dict[str, Dict[str, Dict[str, Any]]]*, *agent_first: bool*, *ignore_keys={}*)

to_numpy() → Dict[str, Dict[str, Dict[str, ndarray]]]

Lossy data transformer, which converts a dict of episode to a dict of numpy array like. (some episode may be empty)

class malib.utils.episode.**NewEpisodeList**(*num: int*, *agents: List[str]*)

Bases: object

record(*data: List[Dict[str, Dict[str, Any]]]*, *agent_first: bool*, *is_episode_done: List[bool]*, *ignore_keys={}*)

to_numpy() → Dict[str, Dict[str, ndarray]]

Lossy data transformer, which converts a dict of episode to a dict of numpy array like. (some episode may be empty)

15.4 malib.utils.exploitability module

class malib.utils.exploitability.**NFSPPolicies**(*game*, *nfsp_policies: List[TabularPolicy]*)

Bases: Policy

Joint policy to be evaluated.

Initializes a policy.

Parameters

- **game** – the game for which this policy applies
- **player_ids** – list of player ids for which this policy applies; each should be in the range 0..game.num_players()-1.

action_probabilities(*state: Any*, *player_id: Optional[str] = None*)

Returns a dictionary {action: prob} for all legal actions.

IMPORTANT: We assume the following properties hold: - All probabilities are >=0 and sum to 1 - TLDR: Policy implementations should list the (action, prob) for all legal

actions, but algorithms should not rely on this (yet). Details: Before May 2020, only legal actions were present in the mapping, but it did not have to be exhaustive: missing actions were considered to be associated to a zero probability. For example, a deterministic state-poliy was previously {action: 1.0}. Given this change of convention is new and hard to enforce, algorithms should not rely on the fact that all legal actions should be present.

Parameters

- **state** – A *pyspiel.State* object.
- **player_id** – Optional, the player id for whom we want an action. Optional unless this is a simultaneous state at which multiple players can act.

Returns
probability}` for the specified player in the supplied state.

Return type
A *dict* of `{action

class malib.utils.exploitability.**OSPolicyWrapper**(*game*, *policy:* Policy, *player_ids: List[int]*, *use_observation*, *tolerance: float = 1e-05*)

Bases: Policy

Initializes a policy.

Parameters

- **game** – the game for which this policy applies
- **player_ids** – list of player ids for which this policy applies; each should be in the range 0..game.num_players()-1.

action_probabilities(*state*, *player_id=None*)

Returns a dictionary {action: prob} for all legal actions.

IMPORTANT: We assume the following properties hold: - All probabilities are >=0 and sum to 1 - TLDR: Policy implementations should list the (action, prob) for all legal

actions, but algorithms should not rely on this (yet). Details: Before May 2020, only legal actions were present in the mapping, but it did not have to be exhaustive: missing actions were considered to be associated to a zero probability. For example, a deterministic state-poliy was previously {action: 1.0}. Given this change of convention is new and hard to enforce, algorithms should not rely on the fact that all legal actions should be present.

Parameters

- **state** – A *pyspiel.State* object.
- **player_id** – Optional, the player id for whom we want an action. Optional unless this is a simultaneous state at which multiple players can act.

Returns
probability}` for the specified player in the supplied state.

Return type
A *dict* of `{action

malib.utils.exploitability.**compute_act_probs**(*game: Game*, *policy:* Policy, *state: State*, *player_id: int*, *use_observation*, *epsilon: float = 1e-05*)

malib.utils.exploitability.**convert_to_os_policies**(*game*, *policies: List[*Policy*]*, *use_observation: bool*, *player_ids: List[int]*) → List[Policy]

malib.utils.exploitability.**measure_exploitability**(*game: Union[str, Game]*, *populations: Dict[str, Dict[str,* Policy*]]*, *policy_mixture_dict: Dict[str, Dict[str, float]]*, *use_observation: bool = False*, *use_cpp_br: bool = False*)

Return a measure of closeness to Nash for a policy in the game. :param game: An open_spiel game, e.g. kuhn_poker. :type game: Union[str, pyspiel.Game] :param populations: A dict of strategy specs, mapping from agent to StrategySpec. :type populations: Dict[AgentID, Dict[PolicyID, Policy]] :param policy_mixture_dict: A dict if policy distribution, maps from agent to a dict of floats. :type policy_mixture_dict: Dict[AgentID, Dict[PolicyID, float]] :param use_cpp_br: Compute best response with C++. Defaults to False. :type use_cpp_br: bool, optional

Returns
An object with the following attributes: - player_improvements: A *[num_players]* numpy array of the improvement for players (i.e. value_player_p_versus_BR - value_player_p). - nash_conv: The sum over all players of the improvements in value that each player could obtain by unilaterally changing their strategy, i.e. sum(player_improvements).

Return type
NashConv

15.5 malib.utils.general module

class malib.utils.general.**BufferDict**

Bases: dict

property capacity: int

index(*indices*)

index_func(*x*, *indices*)

set_data(*index*, *new_data*)

set_data_func(*x*, *index*, *new_data*)

malib.utils.general.**deep_update**(*original: dict*, *new_dict: dict*, *new_keys_allowed: str = False*, *allow_new_subkey_list: Optional[List[str]] = None*, *override_all_if_type_changes: Optional[List[str]] = None*) → dict

Updates original dict with values from new_dict recursively.

If new key is introduced in new_dict, then if new_keys_allowed is not True, an error will be thrown. Further, for sub-dicts, if the key is in the allow_new_subkey_list, then new subkeys can be introduced.

Parameters

- **original** (*dict*) – Dictionary with default values.
- **new_dict** (*dict*) – Dictionary with values to be updated
- **new_keys_allowed** (*bool*) – Whether new keys are allowed.
- **allow_new_subkey_list** (*Optional[List[str]]*) – List of keys that correspond to dict values where new subkeys can be introduced. This is only at the top level.
- **override_all_if_type_changes** (*Optional[List[str]]*) – List of top level keys with value=dict, for which we always simply override the entire value (dict), iff the "type" key in that value dict changes.

malib.utils.general.**flatten_dict**(*dt: Dict*, *delimiter: str = '/'*, *prevent_delimiter: bool = False*, *flatten_list: bool = False*)

Flatten dict.

Output and input are of the same dict type. Input dict remains the same after the operation.

malib.utils.general.**frozen_data**(*data*)

malib.utils.general.**iter_dicts_recursively**(*d1*, *d2*)

Assuming dicts have the exact same structure.

malib.utils.general.**iter_many_dicts_recursively**(**d*, *history=None*)

Assuming dicts have the exact same structure, or raise KeyError.

malib.utils.general.**iterate_recursively**(*d: Dict*)

malib.utils.general.**merge_dicts**(*d1: dict*, *d2: dict*) → dict

Parameters

- **d1** (*dict*) – Dict 1, the original dict template.
- **d2** (*dict*) – Dict 2, the new dict used to udpate.

Returns
A new dict that is d1 and d2 deep merged.

Return type
dict

malib.utils.general.**tensor_cast**(*custom_caster: Optional[Callable] = None*, *callback: Optional[Callable] = None*, *dtype_mapping: Optional[Dict] = None*, *device='cpu'*)

Casting the inputs of a method into tensors if needed.

Note: This function does not support recursive iteration.

Parameters

- **custom_caster** (*Callable, optional*) – Customized caster. Defaults to None.
- **callback** (*Callable, optional*) – Callback function, accepts returns of wrapped function as inputs. Defaults to None.
- **dtype_mapping** (*Dict, optional*) – Specify the data type for inputs which you wanna. Defaults to None.

Returns
A decorator.

Return type
Callable

malib.utils.general.**unflatten_dict**(*dt: Dict[str, T]*, *delimiter: str = '/'*) → Dict[str, T]

Unflatten dict. Does not support unflattening lists.

malib.utils.general.**unflatten_list_dict**(*dt: Dict[str, T]*, *delimiter: str = '/'*) → Dict[str, T]

Unflatten nested dict and list.

This function now has some limitations: (1) The keys of dt must be str. (2) If unflattened dt (the result) contains list, the index order must be

ascending when accessing dt. Otherwise, this function will throw AssertionError.

(3) The unflattened dt (the result) shouldn't contain dict with number keys.

Be careful to use this function. If you want to improve this function, please also improve the unit test. See #14487 for more details.

Parameters

- **dt** (*dict*) – Flattened dictionary that is originally nested by multiple list and dict.
- **delimiter** (*str*) – Delimiter of keys.

Example

```
>>> dt = {"aaa/0/bb": 12, "aaa/1/cc": 56, "aaa/1/dd": 92}
>>> unflatten_list_dict(dt)
{'aaa': [{'bb': 12}, {'cc': 56, 'dd': 92}]}
```

malib.utils.general.**unflattened_lookup**(*flat_key: str*, *lookup: Union[Mapping, Sequence]*, *delimiter: str = '/'*, ***kwargs*) → Union[Mapping, Sequence]

Unflatten *flat_key* and iteratively look up in *lookup*. E.g. *flat_key="a/0/b"* will try to return *lookup["a"][0]["b"]*.

malib.utils.general.**update_configs**(*runtime_config: Dict[str, Any]*)

Update global configs with a given dict

malib.utils.general.**update_dataset_config**(*global_dict: Dict[str, Any]*, *runtime_config: Dict[str, Any]*)

malib.utils.general.**update_evaluation_config**(*global_dict: Dict[str, Any]*, *runtime_config: Dict[str, Any]*)

malib.utils.general.**update_global_evaluator_config**(*global_dict: Dict[str, Any]*, *runtime_config: Dict[str, Any]*)

malib.utils.general.**update_parameter_server_config**(*global_dict: Dict[str, Any]*, *runtime_config: Dict[str, Any]*)

malib.utils.general.**update_rollout_configs**(*global_dict: Dict[str, Any]*, *runtime_dict: Dict[str, Any]*) → Dict[str, Any]

Update default rollout configuration and return a new one.

Note: the keys in rollout configuration include - *num_threads*: int, the total threads in a rollout worker to run simulations. - *num_env_per_thread*: int, indicate how many environment will be created for each running thread. - *batch_mode*: default by 'time_step'. - *post_processor_types*: default by ['default']. - *use_subprov_env*: use sub proc environment or not, default by False. - *num_eval_threads*: the number of threads for evaluation, default by 1.

Parameters

- **global_dict** (*Dict[str, Any]*) – The default global configuration.
- **runtime_dict** (*Dict[str, Any]*) – The default global configuration.

Returns
Updated rollout configuration.

Return type
Dict[str, Any]

malib.utils.general.**update_training_config**(*global_dict: Dict[str, Any]*, *runtime_dict: Dict[str, Any]*) → Dict[str, Any]

15.6 malib.utils.logging module

15.7 malib.utils.monitor module

malib.utils.monitor.**write_to_tensorboard**(*writer: SummaryWriter*, *info: Dict*, *global_step: Union[int, Dict]*, *prefix: str*)

Write learning info to tensorboard.

Parameters

- **writer** (*tensorboard.SummaryWriter*) – The summary writer instance.
- **info** (*Dict*) – The information dict.
- **global_step** (*int*) – The global step indicator.
- **prefix** (*str*) – Prefix added to keys in the info dict.

15.8 malib.utils.notations module

malib.utils.notations.**AGENT_EXPERIENCE_TABLE_NAME_GEN**(*env_id*, *policy_id*, *policy_type*)

malib.utils.notations.**EPISODE_EXPERIENCE_TABLE_NAME_GEN**(*env_id*)

malib.utils.notations.**deprecated**(*func*)

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

15.9 malib.utils.preprocessor module

class malib.utils.preprocessor.**BoxFlattenPreprocessor**(*space: Box*)

Bases: *Preprocessor*

property shape

property size

transform(*data*, *nested=False*) → ndarray

Transform original data to feet the preprocessed shape. Nested works for nested array.

write(*array*, *offset*, *data*)

class malib.utils.preprocessor.**BoxStackedPreprocessor**(*space: Box*)

Bases: *Preprocessor*

property shape

property size

transform(*data*, *nested=False*) → ndarray

Transform original data to feet the preprocessed shape. Nested works for nested array.

write(*array: ndarray*, *offset: int*, *data: Any*)

class malib.utils.preprocessor.**DictFlattenPreprocessor**(*space: Dict*)

Bases: *Preprocessor*

property shape

property size

transform(*data*, *nested=False*) → ndarray

Transform support multi-instance input

write(*array: ndarray*, *offset: int*, *data: Any*)

class malib.utils.preprocessor.**DiscreteFlattenPreprocessor**(*space: Discrete*)

Bases: *Preprocessor*

property shape

property size

transform(*data*, *nested=False*) → ndarray

Transform to one hot

write(*array*, *offset*, *data*)

class malib.utils.preprocessor.**Mode**

Bases: object

FLATTEN = 'flatten'

STACK = 'stack'

class malib.utils.preprocessor.**Preprocessor**(*space: Space*)

Bases: object

property observation_space

property original_space: Space

property shape

property size

abstract transform(*data*, *nested=False*) → ndarray

Transform original data to feet the preprocessed shape. Nested works for nested array.

abstract write(*array: ndarray*, *offset: int*, *data: Any*)

class malib.utils.preprocessor.**TupleFlattenPreprocessor**(*space: Tuple*)

Bases: *Preprocessor*

Init a tuple flatten preprocessor, will stack inner flattend spaces.

Note: All sub spaces in a tuple should be homogeneous.

Parameters

space (*spaces.Tuple*) – A tuple of homogeneous spaces.

property shape

property size

transform(*data*, *nested=False*) → ndarray

Transform original data to feet the preprocessed shape. Nested works for nested array.

write(*array: ndarray*, *offset: int*, *data: Any*)

malib.utils.preprocessor.**get_preprocessor**(*space: Space*, *mode: str = 'flatten'*)

15.10 malib.utils.replay_buffer module

class malib.utils.replay_buffer.**MultiagentReplayBuffer**(*size: int*, *stack_num: int = 1*, *ignore_obs_next: bool = False*, *save_only_last_obs: bool = False*, *sample_avail: bool = False*, ***kwargs*)

Bases: *ReplayBuffer*

add_batch(*data: Dict[str, Dict[str, ndarray]]*)

sample(*batch_size: int*) → Dict[str, Tuple[*Batch*, List[int]]]

class malib.utils.replay_buffer.**ReplayBuffer**(*size: int*, *stack_num: int = 1*, *ignore_obs_next: bool = False*, *save_only_last_obs: bool = False*, *sample_avail: bool = False*, ***kwargs*)

Bases: object

add_batch(*data: Dict[str, ndarray]*)

sample(*batch_size: int*) → Tuple[*Batch*, List[int]]

sample_indices(*batch_size: int*) → Sequence[int]

malib.utils.replay_buffer.**to_numpy**(*x: Any*) → Union[*Batch*, ndarray]

Return an object without torch.Tensor.

15.11 malib.utils.schedules module

This file is used for specifying various schedules that evolve over time throughout the execution of the algorithm, such as:

- learning rate for the optimizer
- exploration epsilon for the epsilon greedy exploration strategy
- beta parameter for beta parameter in prioritized replay

Each schedule has a function *value(t)* which returns the current value of the parameter given the timestep t of the optimization procedure.

Reference: https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/common/schedules.py

class malib.utils.schedules.**ConstantSchedule**(*value*)

Bases: object

Value remains constant over time. :param value: Constant value of the schedule :type value: float

value(*t*)

See Schedule.value

class malib.utils.schedules.**LinearSchedule**(*schedule_timesteps*, *final_p*, *initial_p=1.0*)

Bases: object

Linear interpolation between initial_p and final_p over schedule_timesteps. After this many timesteps pass final_p is returned. :param schedule_timesteps: Number of timesteps for which to linearly anneal initial_p

to final_p

Parameters

- **initial_p** (*float*) – initial output value
- **final_p** (*float*) – final output value

value(*t*)

See Schedule.value

class malib.utils.schedules.**PiecewiseSchedule**(*endpoints*, *interpolation=<function linear_interpolation>*, *outside_value=None*)

Bases: object

Piecewise schedule. endpoints: [(int, int)]

list of pairs *(time, value)* meanining that schedule should output *value* when *t==time*. All the values for time must be sorted in an increasing order. When t is between two times, e.g. *(time_a, value_a)* and *(time_b, value_b)*, such that *time_a <= t < time_b* then value outputs *interpolation(value_a, value_b, alpha)* where alpha is a fraction of time passed between *time_a* and *time_b* for time *t*.

interpolation: lambda float, float, float: float
a function that takes value to the left and to the right of t according to the *endpoints*. Alpha is the fraction of distance from left endpoint to right endpoint that t has covered. See linear_interpolation for example.

outside_value: float
if the value is requested outside of all the intervals sepecified in *endpoints* this value is returned. If None then AssertionError is raised when outside value is requested.

value(*t*)

See Schedule.value

class malib.utils.schedules.**PowerSchedule**(*schedule_timesteps*, *final_p*, *initial_p=1.0*)

Bases: object

value(*t*, *power: int = 1.0*)

class malib.utils.schedules.**Schedule**

Bases: object

value(*t*)

Value of the schedule at time t

malib.utils.schedules.**linear_interpolation**(*l*, *r*, *alpha*)

15.12 malib.utils.statistic module

class malib.utils.statistic.**RunningMeanStd**(*mean: Union[float, ndarray] = 0.0*, *std: Union[float, ndarray] = 1.0*, *clip_max: Optional[float] = 10.0*, *epsilon: float = 1.1920928955078125e-07*)

Bases: object

norm(*data_array: Union[float, ndarray]*) → Union[float, ndarray]

update(*data_array: ndarray*) → None

Add a batch of item into RMS with the same shape, modify mean/var/count.

15.13 malib.utils.stopping_conditions module

class malib.utils.stopping_conditions.**MaxIterationStopping**(*max_iteration: int*)

Bases: *StoppingCondition*

should_stop(*latest_trainer_result: dict*, **args*, ***kwargs*) → bool

class malib.utils.stopping_conditions.**MergeStopping**(*stoppings: List[*StoppingCondition*]*)

Bases: *StoppingCondition*

should_stop(*latest_trainer_result: dict*, **args*, ***kwargs*) → bool

class malib.utils.stopping_conditions.**NoStoppingCondition**

Bases: *StoppingCondition*

should_stop(*latest_trainer_result: dict*, **args*, ***kwargs*) → bool

class malib.utils.stopping_conditions.**RewardImprovementStopping**(*mininum_reward_improvement: float*)

Bases: *StoppingCondition*

should_stop(*latest_trainer_result: dict*, **args*, ***kwargs*) → bool

class malib.utils.stopping_conditions.**StopImmediately**

Bases: *StoppingCondition*

should_stop(*latest_trainer_result: dict*, **args*, ***kwargs*) → bool

class malib.utils.stopping_conditions.**StoppingCondition**

Bases: ABC

abstract should_stop(*latest_trainer_result: dict*, **args*, ***kwargs*) → bool

malib.utils.stopping_conditions.**get_stopper**(*conditions: Dict[str, Any]*)

15.14 malib.utils.tasks_register module

15.15 malib.utils.tianshou_batch module

class malib.utils.tianshou_batch.**Batch**(*batch_dict: Optional[Union[dict,* Batch*, Sequence[Union[dict,* Batch*]], ndarray]] = None*, *copy: bool = False*, ***kwargs: Any*)

Bases: object

The internal data structure in Tianshou.

Batch is a kind of supercharged array (of temporal data) stored individually in a (recursive) dictionary of object that can be either numpy array, torch tensor, or batch themselves. It is designed to make it extremely easily to access, manipulate and set partial view of the heterogeneous data conveniently.

For a detailed description, please refer to batch_concept.

static cat(*batches: Sequence[Union[dict,* Batch*]]*) → *Batch*

Concatenate a list of Batch object into a single new batch.

For keys that are not shared across all batches, batches that do not have these keys will be padded by zeros with appropriate shapes. E.g.

```
>>> a = Batch(a=np.zeros([3, 4]), common=Batch(c=np.zeros([3, 5])))
>>> b = Batch(b=np.zeros([4, 3]), common=Batch(c=np.zeros([4, 5])))
>>> c = Batch.cat([a, b])
>>> c.a.shape
(7, 4)
>>> c.b.shape
(7, 3)
>>> c.common.c.shape
(7, 5)
```

cat_(*batches: Union[Batch, Sequence[Union[dict, Batch]]]*) → None

Concatenate a list of (or one) Batch objects into current batch.

static empty(*batch: Batch*, *index: Optional[Union[slice, int, ndarray, List[int]]] = None*) → *Batch*

Return an empty Batch object with 0 or None filled.

The shape is the same as the given Batch.

empty_(*index: Optional[Union[slice, int, ndarray, List[int]]] = None*) → *Batch*

Return an empty Batch object with 0 or None filled.

If "index" is specified, it will only reset the specific indexed-data.

```
>>> data.empty_()
>>> print(data)
Batch(
    a: array([[0., 0.],
              [0., 0.]]),
    b: array([None, None], dtype=object),
)
>>> b={'c': [2., 'st'], 'd': [1., 0.]}
>>> data = Batch(a=[False,  True], b=b)
>>> data[0] = Batch.empty(data[1])
>>> data
Batch(
    a: array([False,  True]),
    b: Batch(
           c: array([None, 'st']),
           d: array([0., 0.]),
       ),
)
```

is_empty(*recurse: bool = False*) → bool

Test if a Batch is empty.

If `recurse=True`, it further tests the values of the object; else it only tests the existence of any key.

`b.is_empty(recurse=True)` is mainly used to distinguish `Batch(a=Batch(a=Batch()))` and `Batch(a=1)`. They both raise exceptions when applied to `len()`, but the former can be used in `cat`, while the latter is a scalar and cannot be used in `cat`.

Another usage is in `__len__`, where we have to skip checking the length of recursively empty Batch.

```
>>> Batch().is_empty()
True
>>> Batch(a=Batch(), b=Batch(c=Batch())).is_empty()
```

(continues on next page)

(continued from previous page)

```
False
>>> Batch(a=Batch(), b=Batch(c=Batch())).is_empty(recurse=True)
True
>>> Batch(d=1).is_empty()
False
>>> Batch(a=np.float64(1.0)).is_empty()
False
```

property shape: List[int]

Return self.shape.

split(*size: int*, *shuffle: bool = True*, *merge_last: bool = False*) → Iterator[*Batch*]

Split whole data into multiple small batches.

Parameters

- **size** (*int*) – divide the data batch with the given size, but one batch if the length of the batch is smaller than "size".
- **shuffle** (*bool*) – randomly shuffle the entire data batch if it is True, otherwise remain in the same. Default to True.
- **merge_last** (*bool*) – merge the last batch into the previous one. Default to False.

static stack(*batches: Sequence[Union[dict, Batch]]*, *axis: int = 0*) → *Batch*

Stack a list of Batch object into a single new batch.

For keys that are not shared across all batches, batches that do not have these keys will be padded by zeros. E.g.

```
>>> a = Batch(a=np.zeros([4, 4]), common=Batch(c=np.zeros([4, 5])))
>>> b = Batch(b=np.zeros([4, 6]), common=Batch(c=np.zeros([4, 5])))
>>> c = Batch.stack([a, b])
>>> c.a.shape
(2, 4, 4)
>>> c.b.shape
(2, 4, 6)
>>> c.common.c.shape
(2, 4, 5)
```

Note: If there are keys that are not shared across all batches, `stack` with `axis != 0` is undefined, and will cause an exception.

stack_(*batches: Sequence[Union[dict, Batch]]*, *axis: int = 0*) → None

Stack a list of Batch object into current batch.

to_numpy() → None

Change all torch.Tensor to numpy.ndarray in-place.

to_torch(*dtype: Optional[dtype] = None*, *device: Union[str, int, device] = 'cpu'*) → None

Change all numpy.ndarray to torch.Tensor in-place.

update(*batch: Optional[Union[dict, Batch]] = None*, ***kwargs: Any*) → None

Update this batch from another dict/Batch.

15.16 malib.utils.timing module

class malib.utils.timing.**AttrDict**

Bases: dict

class malib.utils.timing.**AvgTime**(*num_values_to_avg*)

Bases: object

tofloat()

class malib.utils.timing.**Timing**

Bases: *AttrDict*

add_time(*key: str*)

Add time additively. :param key: Timer key. :type key: str

Returns
A *TimingContext* instance in additive mode..

Return type
TimingContext

time_avg(*key*, *average=10*)

timeit(*key*)

todict()

class malib.utils.timing.**TimingContext**(*timer*, *key*, *additive=False*, *average=None*)

Bases: object

15.17 malib.utils.typing module

class malib.utils.typing.**BColors**

Bases: object

BOLD = '\x1b[1m'

ENDC = '\x1b[0m'

FAIL = '\x1b[91m'

HEADER = '\x1b[95m'

OKBLUE = '\x1b[94m'

OKCYAN = '\x1b[96m'

OKGREEN = '\x1b[92m'

UNDERLINE = '\x1b[4m'

WARNING = '\x1b[93m'

class malib.utils.typing.**BehaviorMode**(*value*)

Bases: IntEnum

Behavior mode, indicates environment agent behavior

EXPLOITATION = 1

Trigger exploitation mode

EXPLORATION = 0

Trigger exploration mode

class malib.utils.typing.**DataFrame**(*identifier: Any*, *data: Any*, *meta_data: Dict[str, Any]*)

Bases: object

data: Any

identifier: Any

meta_data: Dict[str, Any]

CHAPTER

SIXTEEN

CONTRIBUTING TO MALIB

First and foremost, thanks for taking the time to contribute!

The following is a set of concise guidelines for contributing to MALib and its packages. Feel free to propose changes to this document in a pull request.

16.1 Code of Conduct

- Be respectful to other contributors.
- Keep criticism strictly to code when reviewing pull requests.
- If in doubt about conduct ask the team lead or other contributors.

We encourage all forms of contributions to MALib, not limited to:

- Code review and improvement
- Community events
- Blog posts and promotion of the project
- Feature requests
- Patches
- Test cases

16.2 Setup Development Environment

In addition to following the setup steps in the README, you'll need to install the `[dev]` dependencies.

```
pip install -e .[dev]
```

Once done, you're all set to make your first contribution!

16.3 Where to Get Started

Please take a look at the current open issues to see if any of them interest you. If you are unsure how to get started please take a look at the README.

16.4 Committing

Please take care in using good commit messages as they're useful for debugging, reviewing code, and generally just shows care and quality. *How to Write a Git Commit Message <https://chris.beams.io/posts/git-commit/>* provides a good guideline. At a minimum,

1. Limit the subject line to 50 characters
2. Capitalize the subject line
3. Do not end the subject line with a period
4. Use the imperative mood in the subject line
5. If only changing documentation tag your commit with `[ci skip]`

16.5 Pre-Push Checklist

1. Do your best to see that your code compiles locally.
2. Run `make format`. See [Formatting](#Formatting).
3. Do not push to `main`. Instead make a branch and a *pull request <#submission-of-a-pull-request>*

16.6 Submission of a Pull Request

1. Rebase on master
2. Run `make test` locally to see if all test cases pass
3. If you change platform code, you are responsible to ensure all tests and all examples still run normally.
4. Be sure to include new test cases if introducing a new feature or fixing a bug.
5. Update the documentation and apply comments to the public API. You are encouraged to add usage cases.
6. Update the `CHANGELOG.md` addressing what changes were made for the current version and make sure to indicate the PR # linking the changes.
7. For PR description, describe the problem and add references to the related issues that the request addresses.
8. Request review of your code by at least two other contributors. Try to improve your code as much as possible to lessen the burden on others.
9. Do _not_ keep long living branches. Branches are for a specific task. They should not become a sub repository.
10. After your PR gets approved by at least other contributors you may merge your PR.
11. Please enable squashing on your Pull Request before merging, this helps keep every commit on master in a working state and aids bisecting when searching for regressions.

In the body, give a reason for the pull request and tag in issues that the pull request solves. The `WIP:` is for pull requests that should raise discussion but are not in review state.

You are encouraged to review other people's pull requests and tag in relevant reviewers.

16.7 Communication

16.7.1 Issues

1. Always raise issues in GitLab. Verbal discussion and reports are helpful but _not_ enough. Put things in writing please.
2. Raise specific, single-topic issues. If you find yourself having to use "and" in the issue title, you most likely want to create more than one.

16.7.2 Reporting Bugs

Before reporting a bug please check the list of current issues to see if there are issues already open that match what you are experiencing.

When reporting a bug, include as much info as necessary for reproducing it. If you find a closed issue that appears to be the same problem you are experiencing; please open up a new issue referencing the original issue in the body of the new issue.

Tag the issue as a *bug*.

16.8 Feature Requests

Before requesting a feature please check the list of current issues to see if there is already a feature request similar to yours. Also, make sure that the feature you are requesting is not a bug. If it a bug see [Reporting Bugs](Reporting-Bugs).

Describe as best you can what the feature does and why it is useful. Visual aids help with understanding more complex features.

Tag the issue as a feature request using *enhancement* and if it takes more than a few lines to describe also tag with *discussion*.

16.9 Formatting

16.9.1 Python(Format)

1. Always run `make format` before committing.

The project follows a strict format requirement for python code. We made a decision early on in the project to use *Black <https://github.com/psf/black>*. This makes formatting consistent while eliminating *bike shedding <http://bikeshed.com/>*. If you do not already have it please install it via `pip install black`.

Formatting guarantees that your code will pass the CI formatting test case.

16.9.2 Documentation(Format)

[TODO]

CHAPTER

SEVENTEEN

CHANGELOG

CHAPTER

EIGHTEEN

LICENSE

PYTHON MODULE INDEX

m

malib.agent, 29
malib.agent.agent_interface, 29
malib.agent.async_agent, 32
malib.agent.indepdent_agent, 32
malib.agent.manager, 33
malib.agent.team_agent, 35
malib.backend, 37
malib.backend.offline_dataset_server, 37
malib.backend.parameter_server, 38
malib.common, 41
malib.common.distributions, 41
malib.common.manager, 51
malib.common.strategy_spec, 52
malib.models, 55
malib.models.torch, 55
malib.models.torch.continuous, 55
malib.models.torch.discrete, 59
malib.models.torch.net, 63
malib.remote, 67
malib.remote.interface, 67
malib.rl, 69
malib.rl.a2c, 69
malib.rl.a2c.config, 69
malib.rl.a2c.policy, 69
malib.rl.a2c.trainer, 70
malib.rl.coma, 70
malib.rl.coma.critic, 70
malib.rl.coma.trainer, 71
malib.rl.common, 72
malib.rl.common.misc, 72
malib.rl.common.policy, 73
malib.rl.common.trainer, 75
malib.rl.discrete_sac, 76
malib.rl.discrete_sac.policy, 76
malib.rl.discrete_sac.trainer, 76
malib.rl.dqn, 76
malib.rl.dqn.config, 77
malib.rl.dqn.policy, 77
malib.rl.dqn.trainer, 77
malib.rl.maddpg, 78
malib.rl.maddpg.loss, 78
malib.rl.maddpg.trainer, 78
malib.rl.mappo, 78
malib.rl.mappo.config, 78
malib.rl.mappo.policy, 78
malib.rl.mappo.trainer, 78
malib.rl.pg, 78
malib.rl.pg.config, 78
malib.rl.pg.policy, 78
malib.rl.pg.trainer, 79
malib.rl.ppo, 80
malib.rl.ppo.policy, 80
malib.rl.ppo.trainer, 80
malib.rl.qmix, 80
malib.rl.qmix.q_mixer, 80
malib.rl.qmix.trainer, 80
malib.rl.random, 80
malib.rl.random.config, 80
malib.rl.random.policy, 80
malib.rl.random.random_trainer, 81
malib.rl.sac, 81
malib.rl.sac.policy, 81
malib.rl.sac.trainer, 81
malib.rollout.envs.env, 87
malib.rollout.envs.gym, 83
malib.rollout.envs.gym.env, 83
malib.rollout.envs.mdp, 84
malib.rollout.envs.mdp.env, 84
malib.rollout.envs.open_spiel, 85
malib.rollout.envs.open_spiel.env, 85
malib.rollout.envs.pettingzoo, 86
malib.rollout.envs.pettingzoo.env, 86
malib.rollout.envs.pettingzoo.scenario_configs_ref, 87
malib.rollout.inference, 91
malib.rollout.inference.ray.server, 91
malib.scenarios, 93
malib.scenarios.scenario, 93
malib.utils, 95
malib.utils.data, 95
malib.utils.episode, 95
malib.utils.exploitability, 97
malib.utils.general, 98

malib.utils.logging, 101
malib.utils.monitor, 101
malib.utils.notations, 102
malib.utils.preprocessor, 102
malib.utils.replay_buffer, 104
malib.utils.schedules, 104
malib.utils.statistic, 105
malib.utils.stopping_conditions, 106
malib.utils.tianshou_batch, 106
malib.utils.timing, 109
malib.utils.typing, 109

INDEX

A

A2CPolicy (*class in malib.rl.a2c.policy*), 69
A2CTrainer (*class in malib.rl.a2c.trainer*), 70
ACC_REWARD (*malib.utils.episode.Episode attribute*), 95
ACTION (*malib.utils.episode.Episode attribute*), 95
action_adapter() (*malib.rollout.envs.env.Environment static method*), 87
ACTION_DIST (*malib.utils.episode.Episode attribute*), 95
ACTION_LOGITS (*malib.utils.episode.Episode attribute*), 95
ACTION_MASK (*malib.utils.episode.Episode attribute*), 95
action_mask_extract() (*malib.rollout.envs.env.GroupWrapper method*), 88
action_probabilities() (*malib.utils.exploitability.NFSPPolicies method*), 97
action_probabilities() (*malib.utils.exploitability.OSPolicyWrapper method*), 97
action_spaces (*malib.rollout.envs.env.Environment property*), 87
action_spaces (*malib.rollout.envs.env.GroupWrapper property*), 88
action_spaces (*malib.rollout.envs.env.Wrapper property*), 90
action_spaces (*malib.rollout.envs.gym.env.GymEnv property*), 83
action_spaces (*malib.rollout.envs.mdp.env.MDPEnvironment property*), 84
action_spaces (*malib.rollout.envs.open_spiel.env.OpenSpielEnv property*), 85
action_spaces (*malib.rollout.envs.pettingzoo.env.PettingZooEnv property*), 86
actions_from_params() (*malib.common.distributions.BernoulliDistribution method*), 41
actions_from_params() (*malib.common.distributions.CategoricalDistribution method*), 42
actions_from_params() (*malib.common.distributions.DiagGaussianDistribution method*), 43
actions_from_params() (*malib.common.distributions.Distribution method*), 44
actions_from_params() (*malib.common.distributions.MultiCategoricalDistribution method*), 46
actions_from_params() (*malib.common.distributions.StateDependentNoiseDistribution method*), 48
ActionSpace() (*in module malib.rollout.envs.open_spiel.env*), 85
Actor (*class in malib.models.torch.continuous*), 55
Actor (*class in malib.models.torch.discrete*), 59
actor (*malib.rl.common.policy.Policy property*), 73
ActorCritic (*class in malib.models.torch.net*), 63
ActorProb (*class in malib.models.torch.continuous*), 55
add_batch() (*malib.utils.replay_buffer.MultiagentReplayBuffer method*), 104
add_batch() (*malib.utils.replay_buffer.ReplayBuffer method*), 104
add_policies() (*malib.agent.agent_interface.AgentInterface method*), 30
add_policies() (*malib.agent.manager.TrainingManager method*), 34
add_time() (*malib.utils.timing.Timing method*), 109
ADVANTAGE (*malib.utils.episode.Episode attribute*), 95
AGENT_EXPERIENCE_TABLE_NAME_GEN() (*in module malib.utils.notations*), 102
agent_groups (*malib.agent.manager.TrainingManager property*), 34
agent_groups (*malib.rollout.envs.env.GroupWrapper property*), 88
agent_to_group() (*malib.rollout.envs.env.GroupWrapper method*), 88
AgentInterface (*class in malib.agent.agent_interface*), 29
apply_gradients() (*malib.backend.parameter_server.ParameterServer method*), 38
apply_gradients() (*malib.backend.parameter_server.Table method*), 39
as_remote() (*malib.remote.interface.RemoteInterface*

class method), 67
AsyncAgent (*class in malib.agent.async_agent*), 32
atanh() (*malib.common.distributions.TanhBijector static method*), 50
AttrDict (*class in malib.utils.timing*), 109
AvgTime (*class in malib.utils.timing*), 109

B

Batch (*class in malib.utils.tianshou_batch*), 106
BColors (*class in malib.utils.typing*), 109
BehaviorMode (*class in malib.utils.typing*), 109
BernoulliDistribution (*class in malib.common.distributions*), 41
BOLD (*malib.utils.typing.BColors attribute*), 109
BoxFlattenPreprocessor (*class in malib.utils.preprocessor*), 102
BoxStackedPreprocessor (*class in malib.utils.preprocessor*), 102
BufferDict (*class in malib.utils.general*), 98
build_state_from_observation() (*malib.rollout.envs.env.GroupWrapper method*), 88
build_state_spaces() (*malib.rollout.envs.env.GroupWrapper method*), 89

C

cancel_pending_tasks() (*malib.common.manager.Manager method*), 51
capacity (*malib.utils.general.BufferDict property*), 98
cat() (*malib.utils.tianshou_batch.Batch static method*), 106
cat_() (*malib.utils.tianshou_batch.Batch method*), 107
CategoricalDistribution (*class in malib.common.distributions*), 42
ClientHandler (*class in malib.rollout.inference.ray.server*), 91
close() (*malib.rollout.envs.env.Environment method*), 87
close() (*malib.rollout.envs.env.Wrapper method*), 90
close() (*malib.rollout.envs.gym.env.GymEnv method*), 83
close() (*malib.rollout.envs.mdp.env.MDPEnvironment method*), 84
close() (*malib.rollout.envs.open_spiel.env.OpenSpielEnv method*), 85
close() (*malib.rollout.envs.pettingzoo.env.PettingZooEnv method*), 86
collect_info() (*malib.rollout.envs.env.Environment method*), 87
collect_info() (*malib.rollout.envs.env.Wrapper method*), 90
COMADiscreteCritic (*class in malib.rl.coma.critic*), 70
COMATrainer (*class in malib.rl.coma.trainer*), 71
compute_act_probs() (*in module malib.utils.exploitability*), 98
compute_action() (*malib.rl.common.policy.Policy method*), 73
compute_action() (*malib.rl.dqn.policy.DQNPolicy method*), 77
compute_action() (*malib.rl.pg.policy.PGPolicy method*), 79
compute_action() (*malib.rollout.inference.ray.server.RayInferenceWorke method*), 91
compute_episodic_return() (*malib.utils.data.Postprocessor static method*), 95
connect() (*malib.agent.agent_interface.AgentInterface method*), 30
ConstantSchedule (*class in malib.utils.schedules*), 104
convert_to_os_policies() (*in module malib.utils.exploitability*), 98
coordinate() (*malib.rl.common.policy.Policy method*), 73
copy() (*malib.rl.common.policy.Policy class method*), 73
copy() (*malib.scenarios.scenario.Scenario method*), 93
CosineEmbeddingNetwork (*class in malib.models.torch.discrete*), 59
counter (*malib.rl.common.trainer.Trainer property*), 75
create_joint_action() (*malib.rl.coma.trainer.COMATrainer method*), 71
create_table() (*malib.backend.parameter_server.ParameterServer method*), 38
Critic (*class in malib.models.torch.continuous*), 56
Critic (*class in malib.models.torch.discrete*), 60
critic (*malib.rl.common.policy.Policy property*), 73
CUR_OBS (*malib.utils.episode.Episode attribute*), 95
CUR_STATE (*malib.utils.episode.Episode attribute*), 95
custom_config (*malib.rl.common.policy.Policy property*), 73

D

data (*malib.utils.typing.DataFrame attribute*), 110
DataFrame (*class in malib.utils.typing*), 110
DataParallelNet (*class in malib.models.torch.net*), 63
decode() (*malib.models.torch.continuous.VAE method*), 58
deep_update() (*in module malib.utils.general*), 99
deprecated() (*in module malib.utils.notations*), 102
deregister_state() (*malib.rl.common.policy.Policy method*), 73
device (*malib.agent.agent_interface.AgentInterface property*), 30
device (*malib.rl.common.policy.Policy property*), 73
DiagGaussianDistribution (*class in malib.common.distributions*), 43

DictFlattenPreprocessor (*class in malib.utils.preprocessor*), 102
DiscreteFlattenPreprocessor (*class in malib.utils.preprocessor*), 102
Distribution (*class in malib.common.distributions*), 44
DONE (*malib.utils.episode.Episode attribute*), 96
DQNPolicy (*class in malib.rl.dqn.policy*), 77
DQNTrainer (*class in malib.rl.dqn.trainer*), 77

E

empty() (*malib.utils.tianshou_batch.Batch static method*), 107
empty_() (*malib.utils.tianshou_batch.Batch method*), 107
end_consumer_pipe() (*malib.backend.offline_dataset_server.OfflineDataset method*), 37
end_producer_pipe() (*malib.backend.offline_dataset_server.OfflineDataset method*), 37
ENDC (*malib.utils.typing.BColors attribute*), 109
entropy (*malib.common.distributions.MaskedCategorical property*), 46
entropy() (*malib.common.distributions.BernoulliDistribution method*), 41
entropy() (*malib.common.distributions.CategoricalDistribution method*), 42
entropy() (*malib.common.distributions.DiagGaussianDistribution method*), 43
entropy() (*malib.common.distributions.Distribution method*), 45
entropy() (*malib.common.distributions.MultiCategoricalDistribution method*), 46
entropy() (*malib.common.distributions.SquashedDiagGaussianDistribution method*), 47
entropy() (*malib.common.distributions.StateDependentNoiseDistribution method*), 49
env_desc_gen() (*in module malib.rollout.envs.gym*), 83
env_desc_gen() (*in module malib.rollout.envs.mdp*), 84
env_desc_gen() (*in module malib.rollout.envs.open_spiel*), 85
env_desc_gen() (*in module malib.rollout.envs.pettingzoo*), 86
env_done_check() (*malib.rollout.envs.env.Environment method*), 87
env_done_check() (*malib.rollout.envs.env.GroupWrapper method*), 89
Environment (*class in malib.rollout.envs.env*), 87
Episode (*class in malib.utils.episode*), 95
EPISODE_EXPERIENCE_TABLE_NAME_GEN() (*in module malib.utils.notations*), 102
eps (*malib.rl.dqn.policy.DQNPolicy property*), 77
EXPLOITATION (*malib.utils.typing.BehaviorMode attribute*), 110
EXPLORATION (*malib.utils.typing.BehaviorMode attribute*), 110

F

f() (*malib.models.torch.discrete.NoisyLinear method*), 62
FAIL (*malib.utils.typing.BColors attribute*), 109
FLATTEN (*malib.utils.preprocessor.Mode attribute*), 103
flatten_dict() (*in module malib.utils.general*), 99
force_stop() (*malib.common.manager.Manager method*), 51
forward() (*malib.common.distributions.TanhBijector static method*), 50
forward() (*malib.models.torch.continuous.Actor method*), 55
forward() (*malib.models.torch.continuous.ActorProb method*), 56
forward() (*malib.models.torch.continuous.Critic method*), 57
forward() (*malib.models.torch.continuous.Perturbation method*), 57
forward() (*malib.models.torch.continuous.RecurrentActorProb method*), 57
forward() (*malib.models.torch.continuous.RecurrentCritic method*), 58
forward() (*malib.models.torch.continuous.VAE method*), 58
forward() (*malib.models.torch.discrete.Actor method*), 59
forward() (*malib.models.torch.discrete.CosineEmbeddingNetwork method*), 59
forward() (*malib.models.torch.discrete.Critic method*), 60
forward() (*malib.models.torch.discrete.FractionProposalNetwork method*), 60
forward() (*malib.models.torch.discrete.ImplicitQuantileNetwork method*), 61
forward() (*malib.models.torch.discrete.IntrinsicCuriosityModule method*), 62
forward() (*malib.models.torch.discrete.NoisyLinear method*), 62
forward() (*malib.models.torch.net.DataParallelNet method*), 63
forward() (*malib.models.torch.net.MLP method*), 64
forward() (*malib.models.torch.net.Net method*), 64
forward() (*malib.models.torch.net.Recurrent method*), 64
forward() (*malib.rl.coma.critic.COMADiscreteCritic method*), 70
FractionProposalNetwork (*class in malib.models.torch.discrete*), 60
frozen_data() (*in module malib.utils.general*), 99

G

gae_return() (*malib.utils.data.Postprocessor static method*), 95
gen_policy() (*malib.common.strategy_spec.StrategySpec method*), 52
get_actions() (*malib.common.distributions.Distribution method*), 45
get_algorithm() (*malib.agent.agent_interface.AgentInterface method*), 30
get_algorthms() (*malib.agent.agent_interface.AgentInterface method*), 30
get_exp() (*malib.agent.manager.TrainingManager method*), 34
get_initial_state() (*malib.rl.common.policy.Policy method*), 73
get_interface_state() (*malib.agent.agent_interface.AgentInterface method*), 31
get_meta_data() (*malib.common.strategy_spec.StrategySpec method*), 52
get_noise() (*malib.common.distributions.StateDependentNoiseDistribution method*), 49
get_preprocessor() (*in module malib.utils.preprocessor*), 103
get_std() (*malib.common.distributions.StateDependentNoiseDistribution method*), 49
get_stopper() (*in module malib.utils.stopping_conditions*), 106
get_weights() (*malib.backend.parameter_server.ParameterServer method*), 39
get_weights() (*malib.backend.parameter_server.Table method*), 39
governed_agents (*malib.agent.agent_interface.AgentInterface property*), 31
GroupWrapper (*class in malib.rollout.envs.env*), 88
gumbel_softmax() (*in module malib.rl.common.misc*), 72
GymEnv (*class in malib.rollout.envs.gym.env*), 83

H

HEADER (*malib.utils.typing.BColors attribute*), 109

I

identifier (*malib.utils.typing.DataFrame attribute*), 110
ImplicitQuantileNetwork (*class in malib.models.torch.discrete*), 61
IndependentAgent (*class in malib.agent.indepdent_agent*), 32
index() (*malib.utils.general.BufferDict method*), 98
index_func() (*malib.utils.general.BufferDict method*), 98
INFO (*malib.utils.episode.Episode attribute*), 96
IntrinsicCuriosityModule (*class in malib.models.torch.discrete*), 61
inverse() (*malib.common.distributions.TanhBijector static method*), 50
is_empty() (*malib.utils.tianshou_batch.Batch method*), 107
is_running() (*malib.common.manager.Manager method*), 51
is_running() (*malib.remote.interface.RemoteInterface method*), 67
iter_dicts_recursively() (*in module malib.utils.general*), 99
iter_many_dicts_recursively() (*in module malib.utils.general*), 99
iterate_recursively() (*in module malib.utils.general*), 99

K

kl_divergence() (*in module malib.common.distributions*), 50

L

LAST_REWARD (*malib.utils.episode.Episode attribute*), 96
linear_interpolation() (*in module malib.utils.schedules*), 105
LinearSchedule (*class in malib.utils.schedules*), 104
load() (*malib.rl.common.policy.Policy method*), 73
load() (*malib.rl.dqn.policy.DQNPolicy method*), 77
load_from_checkpoint() (*malib.common.strategy_spec.StrategySpec method*), 52
load_state_dict() (*malib.rl.common.policy.Policy method*), 73
load_state_dict() (*malib.rl.common.policy.SimpleObject method*), 75
log_prob() (*malib.common.distributions.BernoulliDistribution method*), 41
log_prob() (*malib.common.distributions.CategoricalDistribution method*), 42
log_prob() (*malib.common.distributions.DiagGaussianDistribution method*), 43
log_prob() (*malib.common.distributions.Distribution method*), 45
log_prob() (*malib.common.distributions.MaskedCategorical method*), 46
log_prob() (*malib.common.distributions.MultiCategoricalDistribution method*), 46
log_prob() (*malib.common.distributions.SquashedDiagGaussianDistribu method*), 47
log_prob() (*malib.common.distributions.StateDependentNoiseDistributio method*), 49
log_prob_correction() (*malib.common.distributions.TanhBijector method*), 50

log_prob_from_params() (*malib.common.distributions.BernoulliDistribution method*), 41
log_prob_from_params() (*malib.common.distributions.CategoricalDistribution method*), 42
log_prob_from_params() (*malib.common.distributions.DiagGaussianDistribution method*), 44
log_prob_from_params() (*malib.common.distributions.Distribution method*), 45
log_prob_from_params() (*malib.common.distributions.MultiCategoricalDistribution method*), 46
log_prob_from_params() (*malib.common.distributions.SquashedDiagGaussianDistribution method*), 47
log_prob_from_params() (*malib.common.distributions.StateDependentNoiseDistribution method*), 49
logits (*malib.common.distributions.MaskedCategorical property*), 46

M

make_net() (*in module malib.models.torch.net*), 65
make_proba_distribution() (*in module malib.common.distributions*), 51
malib.agent
    module, 29
malib.agent.agent_interface
    module, 29
malib.agent.async_agent
    module, 32
malib.agent.indepdent_agent
    module, 32
malib.agent.manager
    module, 33
malib.agent.team_agent
    module, 35
malib.backend
    module, 37
malib.backend.offline_dataset_server
    module, 37
malib.backend.parameter_server
    module, 38
malib.common
    module, 41
malib.common.distributions
    module, 41
malib.common.manager
    module, 51
malib.common.strategy_spec
    module, 52
malib.models
    module, 55
malib.models.torch
    module, 55
malib.models.torch.continuous
    module, 55
malib.models.torch.discrete
    module, 59
malib.models.torch.net
    module, 63
malib.remote
    module, 67
malib.remote.interface
    module, 67
malib.rl
    module, 69
malib.rl.a2c
    module, 69
malib.rl.a2c.config
    module, 69
malib.rl.a2c.policy
    module, 69
malib.rl.a2c.trainer
    module, 70
malib.rl.coma
    module, 70
malib.rl.coma.critic
    module, 70
malib.rl.coma.trainer
    module, 71
malib.rl.common
    module, 72
malib.rl.common.misc
    module, 72
malib.rl.common.policy
    module, 73
malib.rl.common.trainer
    module, 75
malib.rl.discrete_sac
    module, 76
malib.rl.discrete_sac.policy
    module, 76
malib.rl.discrete_sac.trainer
    module, 76
malib.rl.dqn
    module, 76
malib.rl.dqn.config
    module, 77
malib.rl.dqn.policy
    module, 77
malib.rl.dqn.trainer
    module, 77
malib.rl.maddpg
    module, 78

malib.rl.maddpg.loss
    module, 78
malib.rl.maddpg.trainer
    module, 78
malib.rl.mappo
    module, 78
malib.rl.mappo.config
    module, 78
malib.rl.mappo.policy
    module, 78
malib.rl.mappo.trainer
    module, 78
malib.rl.pg
    module, 78
malib.rl.pg.config
    module, 78
malib.rl.pg.policy
    module, 78
malib.rl.pg.trainer
    module, 79
malib.rl.ppo
    module, 80
malib.rl.ppo.policy
    module, 80
malib.rl.ppo.trainer
    module, 80
malib.rl.qmix
    module, 80
malib.rl.qmix.q_mixer
    module, 80
malib.rl.qmix.trainer
    module, 80
malib.rl.random
    module, 80
malib.rl.random.config
    module, 80
malib.rl.random.policy
    module, 80
malib.rl.random.random_trainer
    module, 81
malib.rl.sac
    module, 81
malib.rl.sac.policy
    module, 81
malib.rl.sac.trainer
    module, 81
malib.rollout.envs.env
    module, 87
malib.rollout.envs.gym
    module, 83
malib.rollout.envs.gym.env
    module, 83
malib.rollout.envs.mdp
    module, 84
malib.rollout.envs.mdp.env
    module, 84
malib.rollout.envs.open_spiel
    module, 85
malib.rollout.envs.open_spiel.env
    module, 85
malib.rollout.envs.pettingzoo
    module, 86
malib.rollout.envs.pettingzoo.env
    module, 86
malib.rollout.envs.pettingzoo.scenario_configs_ref
    module, 87
malib.rollout.inference
    module, 91
malib.rollout.inference.ray.server
    module, 91
malib.scenarios
    module, 93
malib.scenarios.scenario
    module, 93
malib.utils
    module, 95
malib.utils.data
    module, 95
malib.utils.episode
    module, 95
malib.utils.exploitability
    module, 97
malib.utils.general
    module, 98
malib.utils.logging
    module, 101
malib.utils.monitor
    module, 101
malib.utils.notations
    module, 102
malib.utils.preprocessor
    module, 102
malib.utils.replay_buffer
    module, 104
malib.utils.schedules
    module, 104
malib.utils.statistic
    module, 105
malib.utils.stopping_conditions
    module, 106
malib.utils.tianshou_batch
    module, 106
malib.utils.timing
    module, 109
malib.utils.typing
    module, 109
Manager (*class in malib.common.manager*), 51

masked_logits() (*in module malib.rl.common.misc*), 72
masked_softmax() (*malib.common.distributions.MaskedCategorical static method*), 46
MaskedCategorical (*class in malib.common.distributions*), 46
MaxIterationStopping (*class in malib.utils.stopping_conditions*), 106
MDPEnvironment (*class in malib.rollout.envs.mdp.env*), 84
measure_exploitability() (*in module malib.utils.exploitability*), 98
merge_dicts() (*in module malib.utils.general*), 99
MergeStopping (*class in malib.utils.stopping_conditions*), 106
meta_data (*malib.utils.typing.DataFrame attribute*), 110
miniblock() (*in module malib.models.torch.net*), 65
MLP (*class in malib.models.torch.net*), 63
Mode (*class in malib.utils.preprocessor*), 103
mode() (*malib.common.distributions.BernoulliDistribution method*), 41
mode() (*malib.common.distributions.CategoricalDistribution method*), 42
mode() (*malib.common.distributions.DiagGaussianDistribution method*), 44
mode() (*malib.common.distributions.Distribution method*), 45
mode() (*malib.common.distributions.MultiCategoricalDistribution method*), 47
mode() (*malib.common.distributions.SquashedDiagGaussianDistribution method*), 48
mode() (*malib.common.distributions.StateDependentNoiseDistribution method*), 49
model_config (*malib.rl.common.policy.Policy property*), 74
module
    malib.agent, 29
    malib.agent.agent_interface, 29
    malib.agent.async_agent, 32
    malib.agent.indepdent_agent, 32
    malib.agent.manager, 33
    malib.agent.team_agent, 35
    malib.backend, 37
    malib.backend.offline_dataset_server, 37
    malib.backend.parameter_server, 38
    malib.common, 41
    malib.common.distributions, 41
    malib.common.manager, 51
    malib.common.strategy_spec, 52
    malib.models, 55
    malib.models.torch, 55
    malib.models.torch.continuous, 55
    malib.models.torch.discrete, 59
    malib.models.torch.net, 63
    malib.remote, 67
    malib.remote.interface, 67
    malib.rl, 69
    malib.rl.a2c, 69
    malib.rl.a2c.config, 69
    malib.rl.a2c.policy, 69
    malib.rl.a2c.trainer, 70
    malib.rl.coma, 70
    malib.rl.coma.critic, 70
    malib.rl.coma.trainer, 71
    malib.rl.common, 72
    malib.rl.common.misc, 72
    malib.rl.common.policy, 73
    malib.rl.common.trainer, 75
    malib.rl.discrete_sac, 76
    malib.rl.discrete_sac.policy, 76
    malib.rl.discrete_sac.trainer, 76
    malib.rl.dqn, 76
    malib.rl.dqn.config, 77
    malib.rl.dqn.policy, 77
    malib.rl.dqn.trainer, 77
    malib.rl.maddpg, 78
    malib.rl.maddpg.loss, 78
    malib.rl.maddpg.trainer, 78
    malib.rl.mappo, 78
    malib.rl.mappo.config, 78
    malib.rl.mappo.policy, 78
    malib.rl.mappo.trainer, 78
    malib.rl.pg, 78
    malib.rl.pg.config, 78
    malib.rl.pg.policy, 78
    malib.rl.pg.trainer, 79
    malib.rl.ppo, 80
    malib.rl.ppo.policy, 80
    malib.rl.ppo.trainer, 80
    malib.rl.qmix, 80
    malib.rl.qmix.q_mixer, 80
    malib.rl.qmix.trainer, 80
    malib.rl.random, 80
    malib.rl.random.config, 80
    malib.rl.random.policy, 80
    malib.rl.random.random_trainer, 81
    malib.rl.sac, 81
    malib.rl.sac.policy, 81
    malib.rl.sac.trainer, 81
    malib.rollout.envs.env, 87
    malib.rollout.envs.gym, 83
    malib.rollout.envs.gym.env, 83
    malib.rollout.envs.mdp, 84
    malib.rollout.envs.mdp.env, 84
    malib.rollout.envs.open_spiel, 85
    malib.rollout.envs.open_spiel.env, 85
    malib.rollout.envs.pettingzoo, 86
    malib.rollout.envs.pettingzoo.env, 86

malib.rollout.envs.pettingzoo.scenario_configs_ref, 87
malib.rollout.inference, 91
malib.rollout.inference.ray.server, 91
malib.scenarios, 93
malib.scenarios.scenario, 93
malib.utils, 95
malib.utils.data, 95
malib.utils.episode, 95
malib.utils.exploitability, 97
malib.utils.general, 98
malib.utils.logging, 101
malib.utils.monitor, 101
malib.utils.notations, 102
malib.utils.preprocessor, 102
malib.utils.replay_buffer, 104
malib.utils.schedules, 104
malib.utils.statistic, 105
malib.utils.stopping_conditions, 106
malib.utils.tianshou_batch, 106
malib.utils.timing, 109
malib.utils.typing, 109
multiagent_post_process() (*malib.agent.agent_interface.AgentInterface method*), 31
multiagent_post_process() (*malib.agent.indepdent_agent.IndependentAgent method*), 33
multiagent_post_process() (*malib.agent.team_agent.TeamAgent method*), 35
MultiagentReplayBuffer (*class in malib.utils.replay_buffer*), 104
MultiCategoricalDistribution (*class in malib.common.distributions*), 46

N

Net (*class in malib.models.torch.net*), 64
NewEpisodeDict (*class in malib.utils.episode*), 96
NewEpisodeList (*class in malib.utils.episode*), 96
NEXT_ACTION_MASK (*malib.utils.episode.Episode attribute*), 96
NEXT_OBS (*malib.utils.episode.Episode attribute*), 96
NEXT_STATE (*malib.utils.episode.Episode attribute*), 96
NFSPPolicies (*class in malib.utils.exploitability*), 97
NoisyLinear (*class in malib.models.torch.discrete*), 62
norm() (*malib.utils.statistic.RunningMeanStd method*), 105
normalized_entropy (*malib.common.distributions.MaskedCategorical property*), 46
NoStoppingCondition (*class in malib.utils.stopping_conditions*), 106
num_policy (*malib.common.strategy_spec.StrategySpec property*), 52

O

observation_space (*malib.utils.preprocessor.Preprocessor property*), 103
observation_spaces (*malib.rollout.envs.env.Environment property*), 87
observation_spaces (*malib.rollout.envs.env.GroupWrapper property*), 89
observation_spaces (*malib.rollout.envs.env.Wrapper property*), 90
observation_spaces (*malib.rollout.envs.gym.env.GymEnv property*), 83
observation_spaces (*malib.rollout.envs.mdp.env.MDPEnvironment property*), 84
observation_spaces (*malib.rollout.envs.open_spiel.env.OpenSpielEnv property*), 85
observation_spaces (*malib.rollout.envs.pettingzoo.env.PettingZooEnv property*), 86
ObservationSpace() (*in module malib.rollout.envs.open_spiel.env*), 85
OfflineDataset (*class in malib.backend.offline_dataset_server*), 37
OKBLUE (*malib.utils.typing.BColors attribute*), 109
OKCYAN (*malib.utils.typing.BColors attribute*), 109
OKGREEN (*malib.utils.typing.BColors attribute*), 109
onehot_from_logits() (*in module malib.rl.common.misc*), 72
OpenSpielEnv (*class in malib.rollout.envs.open_spiel.env*), 85
original_space (*malib.utils.preprocessor.Preprocessor property*), 103
OSPolicyWrapper (*class in malib.utils.exploitability*), 97

P

parallel_simulate (*malib.rollout.envs.pettingzoo.env.PettingZooEnv property*), 86
parameters() (*malib.rl.common.policy.Policy method*), 74
parameters() (*malib.rl.common.trainer.Trainer method*), 75
parameters() (*malib.rl.dqn.policy.DQNPolicy method*), 77
ParameterServer (*class in malib.backend.parameter_server*), 38
Perturbation (*class in malib.models.torch.continuous*), 57
PettingZooEnv (*class in malib.rollout.envs.pettingzoo.env*), 86
PGPolicy (*class in malib.rl.pg.policy*), 78
PGTrainer (*class in malib.rl.pg.trainer*), 79
PiecewiseSchedule (*class in malib.utils.schedules*), 105
Policy (*class in malib.rl.common.policy*), 73
policy (*malib.rl.common.trainer.Trainer property*), 75

possible_agents (*malib.rollout.envs.env.Environment property*), 87
possible_agents (*malib.rollout.envs.env.GroupWrapper property*), 89
possible_agents (*malib.rollout.envs.env.Wrapper property*), 90
possible_agents (*malib.rollout.envs.gym.env.GymEnv property*), 83
possible_agents (*malib.rollout.envs.mdp.env.MDPEnvironment property*), 84
possible_agents (*malib.rollout.envs.open_spiel.env.OpenSpielEnv property*), 85
possible_agents (*malib.rollout.envs.pettingzoo.env.PettingZooEnv property*), 86
post_process() (*malib.rl.a2c.trainer.A2CTrainer method*), 70
post_process() (*malib.rl.coma.trainer.COMATrainer method*), 71
post_process() (*malib.rl.common.trainer.Trainer method*), 75
post_process() (*malib.rl.dqn.trainer.DQNTrainer method*), 77
post_process() (*malib.rl.pg.trainer.PGTrainer method*), 79
Postprocessor (*class in malib.utils.data*), 95
PowerSchedule (*class in malib.utils.schedules*), 105
PRE_DONE (*malib.utils.episode.Episode attribute*), 96
PRE_REWARD (*malib.utils.episode.Episode attribute*), 96
Preprocessor (*class in malib.utils.preprocessor*), 103
preprocessor (*malib.rl.common.policy.Policy property*), 74
prob() (*malib.common.distributions.CategoricalDistribution method*), 43
prob() (*malib.common.distributions.DiagGaussianDistribution method*), 44
prob() (*malib.common.distributions.Distribution method*), 45
proba_distribution() (*malib.common.distributions.BernoulliDistribution method*), 42
proba_distribution() (*malib.common.distributions.CategoricalDistribution method*), 43
proba_distribution() (*malib.common.distributions.DiagGaussianDistribution method*), 44
proba_distribution() (*malib.common.distributions.Distribution method*), 45
proba_distribution() (*malib.common.distributions.MultiCategoricalDistribution method*), 47
proba_distribution() (*malib.common.distributions.SquashedDiagGaussianDistribution method*), 48
proba_distribution() (*malib.common.distributions.StateDependentNoiseDistribution method*), 49
proba_distribution_net() (*malib.common.distributions.BernoulliDistribution method*), 42
proba_distribution_net() (*malib.common.distributions.CategoricalDistribution method*), 43
proba_distribution_net() (*malib.common.distributions.DiagGaussianDistribution method*), 44
proba_distribution_net() (*malib.common.distributions.Distribution method*), 45
proba_distribution_net() (*malib.common.distributions.MultiCategoricalDistribution method*), 47
proba_distribution_net() (*malib.common.distributions.StateDependentNoiseDistribution method*), 49
probs (*malib.common.distributions.MaskedCategorical property*), 46
pull() (*malib.agent.agent_interface.AgentInterface method*), 31
push() (*malib.agent.agent_interface.AgentInterface method*), 31

R

RandomPolicy (*class in malib.rl.random.policy*), 80
RandomTrainer (*class in malib.rl.random.random_trainer*), 81
RayInferenceWorkerSet (*class in malib.rollout.inference.ray.server*), 91
read_table() (*in module malib.backend.offline_dataset_server*), 38
record() (*malib.utils.episode.Episode method*), 96
record() (*malib.utils.episode.NewEpisodeDict method*), 96
record() (*malib.utils.episode.NewEpisodeList method*), 96
record_episode_info_step() (*malib.rollout.envs.env.Environment method*), 87
record_episode_info_step() (*malib.rollout.envs.env.GroupWrapper method*), 89
Recurrent (*class in malib.models.torch.net*), 64
RecurrentActorProb (*class in malib.models.torch.continuous*), 57
RecurrentCritic (*class in malib.models.torch.continuous*), 58

recver (*malib.rollout.inference.ray.server.ClientHandler property*), 91
register_policy_id() (*malib.common.strategy_spec.StrategySpec method*), 52
register_state() (*malib.rl.common.policy.Policy method*), 74
registered_networks (*malib.rl.common.policy.Policy property*), 74
RemoteInterface (*class in malib.remote.interface*), 67
render() (*malib.rollout.envs.env.Environment method*), 87
render() (*malib.rollout.envs.env.Wrapper method*), 90
render() (*malib.rollout.envs.gym.env.GymEnv method*), 83
render() (*malib.rollout.envs.mdp.env.MDPEnvironment method*), 84
render() (*malib.rollout.envs.pettingzoo.env.PettingZooEnv method*), 86
ReplayBuffer (*class in malib.utils.replay_buffer*), 104
reset() (*malib.agent.agent_interface.AgentInterface method*), 31
reset() (*malib.models.torch.discrete.NoisyLinear method*), 62
reset() (*malib.rl.common.policy.Policy method*), 74
reset() (*malib.rl.common.trainer.Trainer method*), 76
reset() (*malib.rl.dqn.policy.DQNPolicy method*), 77
reset() (*malib.rollout.envs.env.Environment method*), 87
reset() (*malib.rollout.envs.env.GroupWrapper method*), 89
reset() (*malib.rollout.envs.env.Wrapper method*), 90
reset() (*malib.rollout.envs.gym.env.GymEnv method*), 84
reset() (*malib.rollout.envs.mdp.env.MDPEnvironment method*), 84
reset() (*malib.rollout.envs.open_spiel.env.OpenSpielEnv method*), 85
reset() (*malib.rollout.envs.pettingzoo.env.PettingZooEnv method*), 86
retrive_results() (*malib.agent.manager.TrainingManager method*), 34
retrive_results() (*malib.common.manager.Manager method*), 51
REWARD (*malib.utils.episode.Episode attribute*), 96
RewardImprovementStopping (*class in malib.utils.stopping_conditions*), 106
RNN_STATE (*malib.utils.episode.Episode attribute*), 96
rnn_states (*malib.rollout.inference.ray.server.ClientHandler property*), 91
rsample() (*malib.common.distributions.MaskedCategorical method*), 46
run() (*malib.agent.manager.TrainingManager method*), 34
RunningMeanStd (*class in malib.utils.statistic*), 105
runtime_config (*malib.rollout.inference.ray.server.ClientHandler property*), 91
runtime_ids (*malib.agent.manager.TrainingManager property*), 34

S

sample() (*malib.common.distributions.BernoulliDistribution method*), 42
sample() (*malib.common.distributions.CategoricalDistribution method*), 43
sample() (*malib.common.distributions.DiagGaussianDistribution method*), 44
sample() (*malib.common.distributions.Distribution method*), 45
sample() (*malib.common.distributions.MaskedCategorical method*), 46
sample() (*malib.common.distributions.MultiCategoricalDistribution method*), 47
sample() (*malib.common.distributions.SquashedDiagGaussianDistributio method*), 48
sample() (*malib.common.distributions.StateDependentNoiseDistribution method*), 50
sample() (*malib.common.strategy_spec.StrategySpec method*), 53
sample() (*malib.models.torch.discrete.NoisyLinear method*), 62
sample() (*malib.utils.replay_buffer.MultiagentReplayBuffer method*), 104
sample() (*malib.utils.replay_buffer.ReplayBuffer method*), 104
sample_gumbel() (*in module malib.rl.common.misc*), 72
sample_indices() (*malib.utils.replay_buffer.ReplayBuffer method*), 104
sample_noise() (*in module malib.models.torch.discrete*), 62
sample_weights() (*malib.common.distributions.StateDependentNoiseDis method*), 50
save() (*malib.rl.common.policy.Policy method*), 74
save() (*malib.rl.dqn.policy.DQNPolicy method*), 77
save() (*malib.rollout.inference.ray.server.RayInferenceWorkerSet method*), 91
Scenario (*class in malib.scenarios.scenario*), 93
Schedule (*class in malib.utils.schedules*), 105
seed() (*malib.rollout.envs.env.Environment method*), 87
seed() (*malib.rollout.envs.env.Wrapper method*), 90
seed() (*malib.rollout.envs.mdp.env.MDPEnvironment method*), 84
seed() (*malib.rollout.envs.open_spiel.env.OpenSpielEnv method*), 85
seed() (*malib.rollout.envs.pettingzoo.env.PettingZooEnv method*), 86

sender (*malib.rollout.inference.ray.server.ClientHandler property*), 91
set_data() (*malib.utils.general.BufferDict method*), 98
set_data_func() (*malib.utils.general.BufferDict method*), 99
set_running() (*malib.remote.interface.RemoteInterface method*), 67
set_weights() (*malib.backend.parameter_server.ParameterServer method*), 39
set_weights() (*malib.backend.parameter_server.Table method*), 39
setup() (*malib.rl.a2c.trainer.A2CTrainer method*), 70
setup() (*malib.rl.coma.trainer.COMATrainer method*), 71
setup() (*malib.rl.common.trainer.Trainer method*), 76
setup() (*malib.rl.dqn.trainer.DQNTrainer method*), 78
setup() (*malib.rl.pg.trainer.PGTrainer method*), 80
shape (*malib.utils.preprocessor.BoxFlattenPreprocessor property*), 102
shape (*malib.utils.preprocessor.BoxStackedPreprocessor property*), 102
shape (*malib.utils.preprocessor.DictFlattenPreprocessor property*), 102
shape (*malib.utils.preprocessor.DiscreteFlattenPreprocessor property*), 102
shape (*malib.utils.preprocessor.Preprocessor property*), 103
shape (*malib.utils.preprocessor.TupleFlattenPreprocessor property*), 103
shape (*malib.utils.tianshou_batch.Batch property*), 108
should_stop() (*malib.utils.stopping_conditions.MaxIterationStopping method*), 106
should_stop() (*malib.utils.stopping_conditions.MergeStopping method*), 106
should_stop() (*malib.utils.stopping_conditions.NoStoppingCondition method*), 106
should_stop() (*malib.utils.stopping_conditions.RewardImprovementSlopeStopping method*), 106
should_stop() (*malib.utils.stopping_conditions.StopImmediately method*), 106
should_stop() (*malib.utils.stopping_conditions.StoppingCondition method*), 106
shutdown() (*malib.rollout.inference.ray.server.RayInferenceWorkerSet method*), 91
SimpleObject (*class in malib.rl.common.policy*), 75
size (*malib.utils.preprocessor.BoxFlattenPreprocessor property*), 102
size (*malib.utils.preprocessor.BoxStackedPreprocessor property*), 102
size (*malib.utils.preprocessor.DictFlattenPreprocessor property*), 102
size (*malib.utils.preprocessor.DiscreteFlattenPreprocessor property*), 102
size (*malib.utils.preprocessor.Preprocessor property*), 103
size (*malib.utils.preprocessor.TupleFlattenPreprocessor property*), 103
soft_update() (*in module malib.rl.common.misc*), 72
softmax() (*in module malib.rl.common.misc*), 73
split() (*malib.utils.tianshou_batch.Batch method*), 108
SquashedDiagGaussianDistribution (*class in malib.common.distributions*), 47
STACK (*malib.utils.preprocessor.Mode attribute*), 103
stack() (*malib.utils.tianshou_batch.Batch static method*), 108
stack_() (*malib.utils.tianshou_batch.Batch method*), 108
start() (*malib.backend.offline_dataset_server.OfflineDataset method*), 37
start() (*malib.backend.parameter_server.ParameterServer method*), 39
start_consumer_pipe() (*malib.backend.offline_dataset_server.OfflineDataset method*), 37
start_producer_pipe() (*malib.backend.offline_dataset_server.OfflineDataset method*), 38
STATE_ACTION_VALUE (*malib.utils.episode.Episode attribute*), 96
state_dict() (*malib.rl.common.policy.Policy method*), 74
state_dict() (*malib.rl.common.policy.SimpleObject method*), 75
state_spaces (*malib.rollout.envs.env.GroupWrapper property*), 89
STATE_VALUE (*malib.utils.episode.Episode attribute*), 96
STATE_VALUE_TARGET (*malib.utils.episode.Episode attribute*), 96
StateDependentNoiseDistribution (*class in malib.common.distributions*), 48
step() (*malib.rollout.envs.env.Environment method*), 88
step() (*malib.rollout.envs.env.Wrapper method*), 90
step_counter() (*malib.rl.common.trainer.Trainer method*), 76
stop_pending_tasks() (*malib.remote.interface.RemoteInterface method*), 67
StopImmediately (*class in malib.utils.stopping_conditions*), 106
StoppingCondition (*class in malib.utils.stopping_conditions*), 106
StrategySpec (*class in malib.common.strategy_spec*), 52
sum_independent_dims() (*in module malib.common.distributions*), 51
sync_remote_parameters() (*malib.agent.agent_interface.AgentInterface method*), 31

T

Table (*class in malib.backend.parameter_server*), 39
TanhBijector (*class in malib.common.distributions*), 50
target_actor (*malib.rl.common.policy.Policy property*), 74
target_critic (*malib.rl.common.policy.Policy property*), 74
TeamAgent (*class in malib.agent.team_agent*), 35
tensor_cast() (*in module malib.utils.general*), 99
terminate() (*malib.agent.manager.TrainingManager method*), 34
terminate() (*malib.common.manager.Manager method*), 51
time_avg() (*malib.utils.timing.Timing method*), 109
time_step() (*malib.rollout.envs.env.Environment method*), 88
time_step() (*malib.rollout.envs.env.GroupWrapper method*), 89
time_step() (*malib.rollout.envs.gym.env.GymEnv method*), 84
time_step() (*malib.rollout.envs.mdp.env.MDPEnvironment method*), 84
time_step() (*malib.rollout.envs.open_spiel.env.OpenSpielEnv method*), 85
time_step() (*malib.rollout.envs.pettingzoo.env.PettingZooEnv method*), 86
timeit() (*malib.utils.timing.Timing method*), 109
Timing (*class in malib.utils.timing*), 109
TimingContext (*class in malib.utils.timing*), 109
to() (*malib.rl.common.policy.Policy method*), 74
to_numpy() (*in module malib.utils.replay_buffer*), 104
to_numpy() (*malib.utils.episode.Episode method*), 96
to_numpy() (*malib.utils.episode.NewEpisodeDict method*), 96
to_numpy() (*malib.utils.episode.NewEpisodeList method*), 96
to_numpy() (*malib.utils.tianshou_batch.Batch method*), 108
to_torch() (*in module malib.utils.data*), 95
to_torch() (*malib.utils.tianshou_batch.Batch method*), 108
todict() (*malib.utils.timing.Timing method*), 109
tofloat() (*malib.utils.timing.AvgTime method*), 109
train() (*malib.agent.agent_interface.AgentInterface method*), 31
train() (*malib.rl.a2c.trainer.A2CTrainer method*), 70
train() (*malib.rl.coma.trainer.COMATrainer method*), 71
train() (*malib.rl.common.trainer.Trainer method*), 76
train() (*malib.rl.dqn.trainer.DQNTrainer method*), 78
train() (*malib.rl.pg.trainer.PGTrainer method*), 80
train_critic() (*malib.rl.coma.trainer.COMATrainer method*), 71
Trainer (*class in malib.rl.common.trainer*), 75
training (*malib.models.torch.continuous.Actor attribute*), 55
training (*malib.models.torch.continuous.ActorProb attribute*), 56
training (*malib.models.torch.continuous.Critic attribute*), 57
training (*malib.models.torch.continuous.Perturbation attribute*), 57
training (*malib.models.torch.continuous.RecurrentActorProb attribute*), 58
training (*malib.models.torch.continuous.RecurrentCritic attribute*), 58
training (*malib.models.torch.continuous.VAE attribute*), 59
training (*malib.models.torch.discrete.Actor attribute*), 59
training (*malib.models.torch.discrete.CosineEmbeddingNetwork attribute*), 60
training (*malib.models.torch.discrete.Critic attribute*), 60
training (*malib.models.torch.discrete.FractionProposalNetwork attribute*), 61
training (*malib.models.torch.discrete.ImplicitQuantileNetwork attribute*), 61
training (*malib.models.torch.discrete.IntrinsicCuriosityModule attribute*), 62
training (*malib.models.torch.discrete.NoisyLinear attribute*), 62
training (*malib.models.torch.net.ActorCritic attribute*), 63
training (*malib.models.torch.net.DataParallelNet attribute*), 63
training (*malib.models.torch.net.MLP attribute*), 64
training (*malib.models.torch.net.Net attribute*), 64
training (*malib.models.torch.net.Recurrent attribute*), 65
training (*malib.rl.coma.critic.COMADiscreteCritic attribute*), 71
training_config (*malib.rl.common.trainer.Trainer property*), 76
TrainingManager (*class in malib.agent.manager*), 33
transform() (*malib.utils.preprocessor.BoxFlattenPreprocessor method*), 102
transform() (*malib.utils.preprocessor.BoxStackedPreprocessor method*), 102
transform() (*malib.utils.preprocessor.DictFlattenPreprocessor method*), 102
transform() (*malib.utils.preprocessor.DiscreteFlattenPreprocessor method*), 102
transform() (*malib.utils.preprocessor.Preprocessor method*), 103
transform() (*malib.utils.preprocessor.TupleFlattenPreprocessor method*), 103
TupleFlattenPreprocessor (*class in*

malib.utils.preprocessor), 103

U

UNDERLINE (*malib.utils.typing.BColors attribute*), 109
unflatten_dict() (*in module malib.utils.general*), 100
unflatten_list_dict() (*in module malib.utils.general*), 100
unflattened_lookup() (*in module malib.utils.general*), 100
update() (*malib.utils.statistic.RunningMeanStd method*), 105
update() (*malib.utils.tianshou_batch.Batch method*), 108
update_configs() (*in module malib.utils.general*), 100
update_dataset_config() (*in module malib.utils.general*), 100
update_evaluation_config() (*in module malib.utils.general*), 100
update_global_evaluator_config() (*in module malib.utils.general*), 101
update_parameter_server_config() (*in module malib.utils.general*), 101
update_parameters() (*malib.rl.common.policy.Policy method*), 75
update_prob_list() (*malib.common.strategy_spec.StrategySpec method*), 53
update_rollout_configs() (*in module malib.utils.general*), 101
update_training_config() (*in module malib.utils.general*), 101

V

VAE (*class in malib.models.torch.continuous*), 58
validate_meta_data() (*in module malib.common.strategy_spec*), 53
value() (*malib.utils.schedules.ConstantSchedule method*), 104
value() (*malib.utils.schedules.LinearSchedule method*), 105
value() (*malib.utils.schedules.PiecewiseSchedule method*), 105
value() (*malib.utils.schedules.PowerSchedule method*), 105
value() (*malib.utils.schedules.Schedule method*), 105
value_function() (*malib.rl.a2c.policy.A2CPolicy method*), 69
value_function() (*malib.rl.dqn.policy.DQNPolicy method*), 77
value_function() (*malib.rl.pg.policy.PGPolicy method*), 79

W

wait() (*malib.common.manager.Manager method*), 51
WARNING (*malib.utils.typing.BColors attribute*), 109
with_updates() (*malib.scenarios.scenario.Scenario method*), 93
workers (*malib.agent.manager.TrainingManager property*), 34
workers (*malib.common.manager.Manager property*), 52
Wrapper (*class in malib.rollout.envs.env*), 90
write() (*malib.utils.preprocessor.BoxFlattenPreprocessor method*), 102
write() (*malib.utils.preprocessor.BoxStackedPreprocessor method*), 102
write() (*malib.utils.preprocessor.DictFlattenPreprocessor method*), 102
write() (*malib.utils.preprocessor.DiscreteFlattenPreprocessor method*), 103
write() (*malib.utils.preprocessor.Preprocessor method*), 103
write() (*malib.utils.preprocessor.TupleFlattenPreprocessor method*), 103
write_table() (*in module malib.backend.offline_dataset_server*), 38
write_to_tensorboard() (*in module malib.utils.monitor*), 101