# imitation

Center for Human-Compatible AI

Jan 07, 2025

# GETTING STARTED

**Imitation provides clean implementations of imitation and reward learning algorithms**, under a unified and user-friendly API. Currently, we have implementations of Behavioral Cloning, DAgger (with synthetic examples), density-based reward modeling, Maximum Causal Entropy Inverse Reinforcement Learning, Adversarial Inverse Reinforcement Learning, Generative Adversarial Imitation Learning, and Deep RL from Human Preferences.

You can find us on GitHub at http://github.com/HumanCompatibleAI/imitation.

CHAPTER

ONE

# MAIN FEATURES

- Built on and compatible with Stable Baselines 3 (SB3).
- Modular Pytorch implementations of Behavioral Cloning, DAgger, GAIL, and AIRL that can train arbitrary SB3 policies.
- GAIL and AIRL have customizable reward and discriminator networks.
- Scripts to train policies using SB3 and save rollouts from these policies as synthetic “expert” demonstrations.
- Data structures and scripts for loading and storing expert demonstrations.

CHAPTER

# TWO

# CITING IMITATION

If you use `imitation` in your research project, please cite our paper to help us track our impact and enable readers to more easily replicate your results. You may use the following BibTeX:

```
@misc{gleave2022imitation,
  author = {Gleave, Adam and Taufeeque, Mohammad and Rocamonde, Juan and Jenner, Erik␣
↪and Wang, Steven H. and Toyer, Sam and Ernestus, Maximilian and Belrose, Nora and␣
↪Emmons, Scott and Russell, Stuart},
  title = {imitation: Clean Imitation Learning Implementations},
  year = {2022},
  howPublished = {arXiv:2211.11972v1 [cs.LG]},
  archivePrefix = {arXiv},
  eprint = {2211.11972},
  primaryClass = {cs.LG},
  url = {https://arxiv.org/abs/2211.11972},
}
```

## 2.1 What is `imitation`?

`imitation` is an open-source library providing high-quality, reliable and modular implementations of seven reward and imitation learning algorithms, built on modern backends like PyTorch and Stable Baselines3. It includes implementations of *Behavioral Cloning (BC)*, *DAgger*, *Generative Adversarial Imitation Learning (GAIL)*, *Adversarial Inverse Reinforcement Learning (AIRL)*, *Reward Learning through Preference Comparisons*, *Maximum Causal Entropy Inverse Reinforcement Learning (MCE IRL)*, and *Density-based reward modeling*. The algorithms follow a consistent interface, making it simple to train and compare a range of algorithms.

A key use case of `imitation` is as an experimental baseline. Small implementation details in imitation learning algorithms can have significant impacts on performance, which can lead to spurious positive results if a weak experimental baseline is used. To address this challenge, `imitation`'s algorithms have been carefully benchmarked and compared to prior implementations. The codebase is statically type-checked and over 90% of it is covered by automated tests.

In addition to providing reliable baselines, `imitation` aims to simplify the process of developing novel reward and imitation learning algorithms. Its implementations are *modular*: users can freely change the reward or policy network architecture, RL algorithm and optimizer without touching the codebase itself. Algorithms can be extended by subclassing and overriding relevant methods. `imitation` also provides utility methods to handle common tasks to support the development of entirely novel algorithms.

Our goal for `imitation` is to make it easier to use, develop, and compare imitation and reward learning algorithms. The library is in active development, and we welcome contributions and feedback.

Check out our recommended *First Steps* for an overview of how to use the library. We also have tutorials, such as *Train an Agent using Behavior Cloning*, that provide detailed examples of specific algorithms. If you are interested in helping develop `imitation` then we suggest you refer to the *Developer Guide* as well as more specific guidelines for *Contributing*.

## 2.2 Installation

### 2.2.1 Prerequisites

- Python 3.8+
- pip (it helps to make sure this is up-to-date: `pip install -U pip`)
- (on ARM64 Macs) you need to set environment variables due to a bug in grpcio:

```
export GRPC_PYTHON_BUILD_SYSTEM_OPENSSL=1
export GRPC_PYTHON_BUILD_SYSTEM_ZLIB=1
```

- (Optional) OpenGL (to render gym environments)
- (Optional) FFmpeg (to encode videos of renders)

### 2.2.2 Installation from PyPI

To install the latest PyPI release, simply run:

```
pip install imitation
```

### 2.2.3 Installation from source

Installation from source is useful if you wish to contribute to the development of `imitation`, or if you need features that have not yet been made available in a stable release:

```
git clone http://github.com/HumanCompatibleAI/imitation
cd imitation
pip install -e .
```

There are also a number of dependencies used for running tests and building the documentation, which can be installed with:

```
pip install -e ".[dev]"
```

## 2.3 First Steps

Imitation can be used in two main ways: through its command-line interface (CLI) or Python API. The CLI allows you to quickly train and test algorithms and policies directly from the command line. The Python API provides greater flexibility and extensibility, and allows you to inter-operate with your existing Python environment.

### 2.3.1 CLI Quickstart

We provide several CLI scripts as front-ends to the algorithms implemented in `imitation`. These use Sacred for configuration and replicability.

For information on how to configure Sacred CLI options, see the Sacred docs.

```
#!/usr/bin/env bash

# Train PPO agent on pendulum and collect expert demonstrations. Tensorboard logs␣
↪saved in quickstart/rl/
python -m imitation.scripts.train_rl with pendulum environment.fast policy_evaluation.
↪fast rl.fast fast logging.log_dir=quickstart/rl/

# Train GAIL from demonstrations. Tensorboard logs saved in output/ (default log␣
↪directory).
python -m imitation.scripts.train_adversarial gail with pendulum environment.fast␣
↪demonstrations.fast policy_evaluation.fast rl.fast fast demonstrations.
↪path=quickstart/rl/rollouts/final.npz demonstrations.source=local

# Train AIRL from demonstrations. Tensorboard logs saved in output/ (default log␣
↪directory).
python -m imitation.scripts.train_adversarial airl with pendulum environment.fast␣
↪demonstrations.fast policy_evaluation.fast rl.fast fast demonstrations.
↪path=quickstart/rl/rollouts/final.npz demonstrations.source=local
```

**Note:** Remove the `fast` options from the commands above to allow training run to completion.

**Tip:** `python -m imitation.scripts.train_rl print_config` will list Sacred script options. These configuration options are also documented in each script's docstrings.

### 2.3.2 Python Interface Quickstart

Here's an example script that loads CartPole demonstrations and trains BC, GAIL, and AIRL models on that data. You will need to `pip install seals` or `pip install imitation[test]` to run this.

```
"""This is a simple example demonstrating how to clone the behavior of an expert.

Refer to the jupyter notebooks for more detailed examples of how to use the␣
↪algorithms.
"""
import numpy as np
from stable_baselines3 import PPO
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3.ppo import MlpPolicy

from imitation.algorithms import bc
from imitation.data import rollout
from imitation.data.wrappers import RolloutInfoWrapper
from imitation.policies.serialize import load_policy
from imitation.util.util import make_vec_env
```

(continues on next page)

(continued from previous page)

```
rng = np.random.default_rng(0)
env = make_vec_env(
    "seals:seals/CartPole-v0",
    rng=rng,
    post_wrappers=[lambda env, _: RolloutInfoWrapper(env)],  # for computing rollouts
)


def train_expert():
    # note: use `download_expert` instead to download a pretrained, competent expert
    print("Training a expert.")
    expert = PPO(
        policy=MlpPolicy,
        env=env,
        seed=0,
        batch_size=64,
        ent_coef=0.0,
        learning_rate=0.0003,
        n_epochs=10,
        n_steps=64,
    )
    expert.learn(1_000)  # Note: change this to 100_000 to train a decent expert.
    return expert


def download_expert():
    print("Downloading a pretrained expert.")
    expert = load_policy(
        "ppo-huggingface",
        organization="HumanCompatibleAI",
        env_name="seals-CartPole-v0",
        venv=env,
    )
    return expert


def sample_expert_transitions():
    # expert = train_expert()  # uncomment to train your own expert
    expert = download_expert()

    print("Sampling expert transitions.")
    rollouts = rollout.rollout(
        expert,
        env,
        rollout.make_sample_until(min_timesteps=None, min_episodes=50),
        rng=rng,
    )
    return rollout.flatten_trajectories(rollouts)


transitions = sample_expert_transitions()
bc_trainer = bc.BC(
    observation_space=env.observation_space,
    action_space=env.action_space,
    demonstrations=transitions,
    rng=rng,
)
```

(continues on next page)

(continued from previous page)

```
evaluation_env = make_vec_env(
    "seals:seals/CartPole-v0",
    rng=rng,
    env_make_kwargs={"render_mode": "human"},  # for rendering
)

print("Evaluating the untrained policy.")
reward, _ = evaluate_policy(
    bc_trainer.policy,  # type: ignore[arg-type]
    evaluation_env,
    n_eval_episodes=3,
    render=True,  # comment out to speed up
)
print(f"Reward before training: {reward}")

print("Training a policy using Behavior Cloning")
bc_trainer.train(n_epochs=1)

print("Evaluating the trained policy.")
reward, _ = evaluate_policy(
    bc_trainer.policy,  # type: ignore[arg-type]
    evaluation_env,
    n_eval_episodes=3,
    render=True,  # comment out to speed up
)
print(f"Reward after training: {reward}")
```

## 2.4 Command Line Interface

Many features of the core library are accessible via the command line interface built using the Sacred package.

Sacred is used to configure and run the algorithms. It is centered around the concept of experiments which are composed of reusable ingredients. Each experiment and each ingredient has its own configuration namespace. Named configurations are used to specify a coherent set of configuration values. It is recommended to at least read the Sacred documentation about the command line interface.

The *scripts* package contains a number of sacred experiments to either execute algorithms or perform utility tasks. The most important *ingredients* for imitation learning are:

- *Environments*
- *Expert Policies*
- *Expert Demonstrations*
- *Reward Functions*

## 2.4.1 Usage Examples

Here we demonstrate some usage examples for the command line interface. You can always find out all the configurable values by running:

```
python -m imitation.scripts.<script> print_config
```

### Run BC on the `CartPole-v1` environment with a pre-trained PPO policy as expert

**Note:** Here the cartpole environment is specified via a named configuration.

```
python -m imitation.scripts.train_imitation bc with \
    cartpole \
    demonstrations.n_expert_demos=50 \
    bc.train_kwargs.n_batches=2000 \
    expert.policy_type=ppo \
    expert.loader_kwargs.path=tests/testdata/expert_models/cartpole_0/policies/final/
↪model.zip
```

50 expert demonstrations are sampled from the PPO policy that is included in the testdata folder. 2000 batches are enough to train a good policy.

### Run DAgger on the `CartPole-v0` environment with a random policy as expert

```
python -m imitation.scripts.train_imitation dagger with \
    cartpole \
    dagger.total_timesteps=2000 \
    demonstrations.n_expert_demos=10 \
    expert.policy_type=random
```

This will not produce any meaningful results, since a random policy is not a good expert.

### Run AIRL on the `MountainCar-v0` environment with a expert from the HuggingFace model hub

```
python -m imitation.scripts.train_adversarial airl with \
    seals_mountain_car \
    total_timesteps=5000 \
    expert.policy_type=ppo-huggingface \
    demonstrations.n_expert_demos=500
```

**Note:** The small number of total timesteps is only for demonstration purposes and will not produce a good policy.

### Run GAIL on the `seals/Swimmer-v0` environment

Here we do not use the named configuration for the seals environment, but instead specify the gym_id directly. The `seals:` prefix ensures that the seals package is imported and the environment is registered.

**Note:** The Swimmer environment needs *mujoco_py* to be installed.

```
python -m imitation.scripts.train_adversarial gail with \
        environment.gym_id="seals:seals/Swimmer-v0" \
        total_timesteps=5000 \
        demonstrations.n_expert_demos=50
```

### Train an expert and save the rollouts explicitly, then train a policy on the saved rollouts

First, train an expert and save the demonstrations. By default, this will use `PPO` and train for 1M time steps. We can set the number of time steps to train for by setting `total_timesteps`. After training the expert, we generate rollouts using the expert policy and save them to disk. We can set a minimum number of episodes or time steps to be saved by setting one of `rollout_save_n_episodes` or `rollout_save_n_timesteps`. Note that the number of episodes or time steps saved may be slightly larger than the specified number.

By default the demonstrations are saved in `<log_dir>/rollouts/final` (where for this script by default `<log_dir>` is `output/train_rl/<environment>/<timestamp>`). However, we can pass an explicit path as logging directory.

```
python -m imitation.scripts.train_rl with seals_cartpole \
        total_timesteps=40000 \
        logging.log_dir=output/ppo/seals_cartpole/trained \
        rollout_save_n_episodes=50
```

Instead of training a new expert, we can also load a pre-trained expert policy and generate rollouts from it. This can be achieved using the `eval_policy` script.

Note that the rollout_save_path is relative to the `log_dir` of the imitation script.

```
python -m imitation.scripts.eval_policy with seals_cartpole \
        expert.policy_type=ppo-huggingface \
        eval_n_episodes=50 \
        logging.log_dir=output/ppo/seals_cartpole/loaded \
        rollout_save_path=rollouts/final
```

Now we can run the imitation script (in this case DAgger) and pass the path to the demonstrations we just generated

```
python -m imitation.scripts.train_imitation dagger with \
        seals_cartpole \
        dagger.total_timesteps=2000 \
        demonstrations.source=local \
        demonstrations.path=output/ppo/seals_cartpole/loaded/rollouts/final
```

### Visualise saved policies

We can use the `eval_policy` script to visualise and render a saved policy. Here we are looking at the policy saved by the previous example.

```
python -m imitation.scripts.eval_policy with \
        expert.policy_type=ppo \
        expert.loader_kwargs.path=output/train_rl/Pendulum-v1/my_run/policies/final/
↪model.zip \
        environment.num_vec=1 \
        render=True \
        environment.gym_id='Pendulum-v1'
```

### Comparing algorithms' performance

Let's use the CLI to compare the performance of different algorithms.

First, let's train an expert on the `CartPole-v1` environment.

```
python -m imitation.scripts.train_rl with \
        cartpole \
        logging.log_dir=output/train_rl/CartPole-v1/expert \
        total_timesteps=10000
```

Now let's train a weaker agent.

```
python -m imitation.scripts.train_rl with \
    cartpole \
    logging.log_dir=output/train_rl/CartPole-v1/non_expert \
    total_timesteps=1000     # simply training less
```

We can evaluate each policy using the `eval_policy` script. For the expert:

```
python -m imitation.scripts.eval_policy with \
        expert.policy_type=ppo \
        expert.loader_kwargs.path=output/train_rl/CartPole-v1/expert/policies/final/
↪model.zip \
        environment.gym_id='CartPole-v1' \
        environment.num_vec=1 \
        logging.log_dir=output/eval_policy/CartPole-v1/expert
```

which will return something like

```
INFO - eval_policy - Result: {
        'n_traj': 74,
        'monitor_return_len': 74,
        'return_min': 26.0,
        'return_mean': 154.21621621621622,
        'return_std': 79.94377589657559,
        'return_max': 500.0,
        'len_min': 26,
        'len_mean': 154.21621621621622,
        'len_std': 79.94377589657559,
        'len_max': 500,
        'monitor_return_min': 26.0,
        'monitor_return_mean': 154.21621621621622,
```

(continues on next page)

(continued from previous page)

```
        'monitor_return_std': 79.94377589657559,
        'monitor_return_max': 500.0
    }
INFO - eval_policy - Completed after 0:00:12
```

For the non-expert:

```
python -m imitation.scripts.eval_policy with \
        expert.policy_type=ppo \
        expert.loader_kwargs.path=output/train_rl/CartPole-v1/non_expert/policies/
↪final/model.zip \
        environment.gym_id='CartPole-v1' \
        environment.num_vec=1 \
        logging.log_dir=output/eval_policy/CartPole-v1/non_expert
```

```
INFO - eval_policy - Result: {
        'n_traj': 355,
        'monitor_return_len': 355,
        'return_min': 8.0,
        'return_mean': 28.92676056338028,
        'return_std': 15.686012049373561,
        'return_max': 104.0,
        'len_min': 8,
        'len_mean': 28.92676056338028,
        'len_std': 15.686012049373561,
        'len_max': 104,
        'monitor_return_min': 8.0,
        'monitor_return_mean': 28.92676056338028,
        'monitor_return_std': 15.686012049373561,
        'monitor_return_max': 104.0
}
INFO - eval_policy - Completed after 0:00:17
```

This will save the monitor CSVs (one for each vectorised env, controlled by environment.num_vec). The monitor CSVs follow the naming convention `mon*.monitor.csv`. We can load these CSV files with `pandas` and use the `imitation.test.reward_improvement` module to compare the performances of the two policies.

```python
from pathlib import Path
import pandas as pd
from imitation.testing.reward_improvement import is_significant_reward_improvement

expert_monitor = pd.concat(
    [
        pd.read_csv(f, skiprows=1)
        for f in Path("./output/train_rl/CartPole-v1/expert/monitor").glob(
            "mon*.monitor.csv"
        )
    ]
)
non_expert_monitor = pd.concat(
    [
        pd.read_csv(f, skiprows=1)
        for f in Path("./output/train_rl/CartPole-v1/non_expert/monitor").glob(
            "mon*.monitor.csv"
        )
    ]
```

(continues on next page)

(continued from previous page)

```
)
if is_significant_reward_improvement(non_expert_monitor["r"], expert_monitor["r"], 0.
↪05):
    print("The expert improved over the non-expert with >95% probability")
else:
    print("No significant (p=0.05) reward improvement of expert over non-expert")
```

```
True
```

### 2.4.2 Algorithm Scripts

Call the algorithm scripts like this:

```
python -m imitation.scripts.<script> [command] with <named_config> <config_values>
```

| algorithm | script | command |
|---|---|---|
| BC | train_imitation | bc |
| DAgger | train_imitation | dagger |
| AIRL | train_adversarial | airl |
| GAIL | train_adversarial | gail |
| Preference Comparison | train_preference_comparisons | • |
| MCE IRL | none | • |
| Density Based Reward Estimation | none | • |

### 2.4.3 Utility Scripts

Call the utility scripts like this:

```
python -m imitation.scripts.<script>
```

| Functionality | Script |
|---|---|
| Reinforcement Learning | *train_rl* |
| Evaluating a Policy | *eval_policy* |
| Parallel Execution of Algorithm Scripts | *parallel* |
| Converting Trajectory Formats | *convert_trajs* |
| Analyzing Experimental Results | *analyze* |

### 2.4.4 Output Directories

The results of the script runs are stored in the following directory structure:

```
output
├── <algo>
│   └── <environment>
│       └── <timestamp>
│           ├── log
│           ├── monitor
│           └── sacred -> ../../../sacred/<script_name>/1
└── sacred
    └── <script_name>
        ├── 1
        └── _sources
```

It contains the final model, tensorboard logs, sacred logs and the sacred source files.

## 2.5 Experts

The algorithms in the imitation library are all about learning from some kind of expert. In many cases this expert is a piece of software itself. The *imitation* library natively supports experts trained using the stable-baselines3 reinforcement learning library.

For example, BC and DAgger can learn from an expert policy and the command line interface of AIRL/GAIL allows one to specify an expert to sample demonstrations from.

In the *First Steps* tutorial, we first train an expert policy using the stable-baselines3 library and then imitate it's behavior using *Behavioral Cloning (BC)*. In practice, you may want to load a pre-trained policy for performance reasons.

### 2.5.1 Loading a policy from a file

The Python interface provides a `load_policy()` function to which you pass a *policy_type*, a VecEnv and any extra kwargs to pass to the corresponding policy loader.

```python
import numpy as np
from imitation.policies.serialize import load_policy
from imitation.util import util

venv = util.make_vec_env("your-env", n_envs=4, rng=np.random.default_rng())
local_policy = load_policy("ppo", venv, path="path/to/model.zip")
```

To load a policy from disk, use either *ppo* or *sac* as the policy type. The path is specified by *path* in the *loader_kwargs* and it should either point to a zip file containing the policy or a directory containing a *model.zip* file that was created by stable-baselines3.

In the command line interface the *expert.policy_type* and *expert.loader_kwargs* parameters define the expert policy to load. For example, to train AIRL on a PPO expert, you would use the following command:

```
python -m imitation.scripts.train_adversarial airl \
    with expert.policy_type=ppo expert.loader_kwargs.path="path/to/model.zip"
```

### 2.5.2 Loading a policy from HuggingFace

HuggingFace is a popular repository for pre-trained models.

To load a stable-baselines3 policy from HuggingFace, use either *ppo-huggingface* or *sac-huggingface* as the policy type. By default, the policies are loaded from the HumanCompatibleAI organization, but you can override this by setting the *organization* parameter in the *loader_kwargs*. When using the Python API, you also have to specify the environment name as *env_name*.

```python
import numpy as np
from imitation.policies.serialize import load_policy
from imitation.util import util

venv = util.make_vec_env("your-env", n_envs=4, rng=np.random.default_rng())
remote_policy = load_policy(
    "ppo-huggingface",
    organization="your-org",
    env_name="your-env",
    venv=venv,
    )
)
```

In the command line interface, the *env-name* is automatically injected into the *loader_kwargs* and does not need to be defined explicitly. In this example, to train AIRL on a PPO expert that was loaded from *your-org* on HuggingFace:

```
python -m imitation.scripts.train_adversarial airl \
    with expert.policy_type=ppo-huggingface expert.loader_kwargs.organization=your-org
```

### 2.5.3 Uploading a policy to HuggingFace

The huggingface-sb3 package provides utilities to push your models to HuggingFace and load them from there. Make sure to use the naming scheme helpers as described in the readme. Otherwise, the loader will not be able to find your model in the repository.

For a convenient high-level interface to train RL models and upload them to HuggingFace, we recommend using the rl-baselines3-zoo.

### 2.5.4 Custom expert types

If you want to use a custom expert type, you can write a corresponding factory function according to `PolicyLoaderFn()` and then register it at the `policy_registry`. For example:

```python
from imitation.policies.serialize import policy_registry
from stable_baselines3.common import policies

def my_policy_loader(venv, some_param: int) -> policies.BasePolicy:
    # load your policy here
    return policy

policy_registry.register("my-policy", my_policy_loader)
```

Then, you can use *my-policy* as the *policy_type* in the command line interface or the Python API:

```
python -m imitation.scripts.train_adversarial airl \
    with expert.policy_type=my-policy expert.loader_kwargs.some_param=42
```

## 2.6 Trajectories

For imitation learning we need trajectories. Trajectories are sequences of observations and actions and sometimes rewards, which are generated by an agent interacting with an environment. They are also called rollouts or episodes. Some are generated by experts and serve as demonstrations, others are generated by the agent and serve as training data for a discriminator. In this library they are stored in a `Trajectory` dataclass:

```
@dataclasses.dataclass(frozen=True)
class Trajectory:
    obs: np.ndarray
    """Observations, shape (trajectory_len + 1, ) + observation_shape."""

    acts: np.ndarray
    """Actions, shape (trajectory_len, ) + action_shape."""

    infos: Optional[np.ndarray]
        """An array of info dicts, shape (trajectory_len, )."""

    terminal: bool
    """Does this trajectory (fragment) end in a terminal state?"""
```

The info dictionaries are optional and can contain arbitrary information. Look at the `Trajectory` class as well as the gymnasium documentation for more details. `TrajectoryWithRew` is a subclass of `Trajectory` and has another `rews` field, which is an array of rewards of shape *(trajectory_len, )*.

Usually, they are passed around as sequences of trajectories.

Some algorithms do not need as much information about the ordering of states, actions and rewards. Rather than using trajectories, these algorithms can make use of individual `Transitions` (`flattened` trajectories).

### 2.6.1 Generating Trajectories

To generate trajectories from a given policy, run the following command:

```
import numpy as np
import imitation.data.rollout as rollout

your_trajectories = rollout.rollout(
    your_policy,
    your_env,
    sample_until=rollout.make_sample_until(min_episodes=10),
    rng=np.random.default_rng(),
    unwrap=False,
)
```

### 2.6.2 Storing/Loading Trajectories

Trajectories can be stored on disk or uploaded to the HuggingFace Dataset Hub.

This will store the sequence of trajectories into a directory at *your_path* as a HuggingFace Dataset:

```
from imitation.data import serialize
serialize.save(your_path, your_trajectories)
```

In the same way you can load trajectories from a HuggingFace Dataset:

```
from imitation.data import serialize
your_trajectories = serialize.load(your_path)
```

Note that some older, now deprecated, trajectory formats are supported by `this loader`, but not by the `saver`.

### 2.6.3 Sharing Trajectories with the HuggingFace Dataset Hub

To share your trajectories with the HuggingFace Dataset Hub, you need to create a HuggingFace account and log in with the HuggingFace CLI:

```
$ huggingface-cli login
```

Then you can upload your trajectories to the HuggingFace Dataset Hub:

```
from imitation.data.huggingface_utils import trajectories_to_dataset

trajectories_to_dataset(your_trajectories).push_to_hub("your_hf_name/your_dataset_name
↪")
```

To use a public dataset from the HuggingFace Dataset Hub, you can use the following code:

```
import datasets
from imitation.data.huggingface_utils import TrajectoryDatasetSequence

your_dataset = datasets.load_dataset("your_hf_name/your_dataset_name")
your_trajectories = TrajectoryDatasetSequence(your_dataset["train"])
```

The `TrajectoryDatasetSequence` wraps a HuggingFace dataset so it can be used in the same way as a list of trajectories.

For example, you can analyze the dataset with `imitation.data.rollout.rollout_stats()` to get the mean return:

```
from imitation.data.rollout import rollout_stats

stats = rollout_stats(your_trajectories)
print(stats["return_mean"])
```

## 2.7 Reward Networks

The goal of both inverse reinforcement learning (IRL) algorithms (e.g. *AIRL*, *GAIL*) and *preference comparison* is to discover a reward function. In imitation learning, these discovered rewards are parameterized by reward networks.

### 2.7.1 Reward Network API

Reward networks need to support two separate but equally important modes of operation. First, these networks need to produce a reward that can be differentiated and used for training the reward network. These rewards are provided by the *forward* method. Second, these networks need to produce a reward that can be used for training policies. These rewards are provided by the *predict_processed* method, which applies additional post-processing that is unhelpful during reward network training.

### 2.7.2 Reward Network Architecture

In imitation learning, reward networks are torch.nn.Module. Out of the box, imitation provides a few reward network architectures such as multi-layer perceptron *BasicRewardNet* and a convolutional neural net CNNRewardNet. To implement your own custom reward network, you can subclass *RewardNet*.

```
from imitation.rewards.reward_nets import RewardNet
import torch as th

class MyRewardNet(RewardNet):
    def __init__(self, observation_space, action_space):
        super().__init__(observation_space, action_space)
        # initialize your custom reward network here

    def forward(self,
        state: th.Tensor, # (batch_size, *obs_shape)
        action: th.Tensor, # (batch_size, *action_shape)
        next_state: th.Tensor, # (batch_size, *obs_shape)
        done: th.Tensor, # (batch_size,)
    ) -> th.Tensor:
        # implement your custom reward network here
        return th.zeros_like(done) # (batch_size,)
```

### 2.7.3 Replace an Environment's Reward with a Reward Network

In order to use a reward network to train a policy, we need to integrate it into an environment. This is done by wrapping the environment in a *RewardVecEnvWrapper*. This wrapper replaces the environment's reward function with the reward network's function.

```
from imitation.util import util
from imitation.rewards.reward_wrapper import RewardVecEnvWrapper
from imitation.rewards.reward_nets import BasicRewardNet

reward_net = BasicRewardNet(obs_space, action_space)
venv = util.make_vec_env("Pendulum-v1", n_envs=3, rng=rng)
venv = RewardVecEnvWrapper(venv, reward_net.predict_processed)
```

### 2.7.4 Reward Network Wrappers

Imitation learning algorithms should converge to a reward function that will theoretically induce the optimal or soft-optimal policy. However, these reward functions may not always be well suited for training RL agents, or we may want to modify them to encourage exploration, for instance.

There are two types of wrapper:

- *ForwardWrapper* allows for direct modification of the results of the reward network's `forward` method. It is used during the learning of the reward network and thus must be differentiable. These wrappers are always applied first and are thus take effect regardless of weather you call *forward*, *predict* or *predict_processed*. They are used for applying transformations like potential shaping (see `ShapedRewardNet`).
- *PredictProcessedWrapper* modifies the predict_processed call to the reward network. Thus this type of reward network wrapper is designed to only modify the reward when it is being used to train/evaluate a policy but *not* when we are taking gradients on it. Thus it does not have to be differentiable.

The most commonly used is the `NormalizedRewardNet` which is a predict procssed wrapper. This class uses a normalization layer to standardize the *output* of the reward function using its running mean and variance, which is useful for stabilizing training. When a reward network is saved, its wrappers are saved along with it, so that the normalization fit during reward learning can be used during future policy learning or evaluation.

```
from imitation.rewards.reward_nets import NormalizedRewardNet
from imitation.util.networks import RunningNorm
train_reward_net = NormalizedRewardNet(
    reward_net,
    normalize_output_layer=RunningNorm,
)
```

**Note:** The reward normalization wrapper does _not_ function identically to stable baselines3's VecNormalize environment wrapper. First, it does not normalize the observations. Second, unlike `VecNormalize`, it scales and centers the reward using the base rewards's mean and variance. The `VecNormalizes` scales the reward down using a running estimate of the _return_.

By default, the normalization wrapper updates the normalization on each call to `predict_processed`. This behavior can be altered as shown below.

```
from functools import partial
eval_rew_fn = partial(reward_net.predict_processed, update_stats=False)
```

### 2.7.5 Serializing and Deserializing Reward Networks

Reward networks, wrappers included, are serialized simply by calling `th.save(reward_net, path)`.

However, when evaluating reward networks, we may or may not want to include the wrappers it was trained with. To load the reward network just as it was saved, wrappers included, we can simply call `th.load(path)`. When using a learned reward network to train or evaluate a policy, we can select whether or not to include the reward network wrappers and convert it into a function using the *load_reward* utility. For example, we might want to remove or keep the reward normalization fit during training in the evaluation phase.

```
import torch as th
from imitation.rewards.serialize import load_reward
from imitation.rewards.reward_nets import NormalizedRewardNet

```

(continues on next page)

(continued from previous page)

```
th.save(train_reward_net, path)
train_reward_net = th.load(path)
# We can also load the reward network as a reward function for use in evaluation
eval_rew_fn_normalized = load_reward(reward_type="RewardNet_normalized", reward_
↪path=path, venv=venv)
eval_rew_fn_unnormalized = load_reward(reward_type="RewardNet_unnormalized", reward_
↪path=path, venv=venv)
# If we want to continue to update the reward networks normalization by default it is␣
↪frozen for evaluation and retraining
rew_fn_normalized = load_reward(reward_type="RewardNet_normalized", reward_path=path,␣
↪venv=venv, update_stats=True)
```

## 2.8 Limitations on Horizon Length

**Warning:** Variable Horizon Environments Considered Harmful

Reinforcement learning (RL) algorithms are commonly trained and evaluated in *variable horizon* environments. In these environments, the episode ends when some termination condition is reached (rather than after a fixed number of steps). This typically corresponds to success, such as reaching the top of the mountain in `MountainCar`, or to failure, such as the pole falling down in `CartPole`. A variable horizon will tend to speed up RL training, by increasing the proportion of samples where the agent's actions still have a meaningful impact on the reward, pruning out states that are already a foregone conclusion.

However, termination conditions must be carefully hand-designed for each environment. Their inclusion therefore provides a significant source of information about the reward. Evaluating reward and imitation learning algorithms in variable-horizon environments can therefore be deeply misleading. In fact, reward learning in commonly used variable horizon environments such as `MountainCar` and `CartPole` can be solved by learning a single bit: the sign of the reward. Of course, an algorithm being able to learn a single bit predicts very little about its performance in real-world tasks, that do not usually come with such an informative termination condition.

To make matters worse, some algorithms have a strong inductive bias towards a particular sign. Indeed, Figure 5 of Kostrikov et al (2021) shows that GAIL is able to reach a third of expert performance even without seeing any expert demonstrations. Consequently, algorithms that happen to have an inductive bias aligned with the task (e.g. positive reward bias for environments where longer episodes are better) may outperform unbiased algorithms on certain environments. Conversely, algorithms with a misaligned inductive bias will perform worse than an unbiased algorithm. This may lead to illusory discrepancies between algorithms, or even different implementations of the same algorithm.

Kostrikov et al (2021) introduces a way to correct for this bias. However, this does not solve the problem of information leakage. Rather, it merely ensures that different algorithms are all able to equally exploit the information leak provided by the termination condition.

In light of this issue, we would strongly recommend users evaluate `imitation` and other reward or imitation learning algorithms only in fixed-horizon environments. This is a common, though unfortunately not ubiquitous, practice in reward learning papers. For example, Christiano et al (2017) use fixed horizon environments because:

> Removing variable length episodes leaves the agent with only the information encoded in the environment itself; human feedback provides its only guidance about what it ought to do.

Many environments, like `HalfCheetah`, are naturally fixed-horizon. Moreover, most variable-horizon tasks can be easily converted into fixed-horizon tasks. Our sister project seals provides fixed-horizon versions of many commonly used MuJoCo continuous control tasks, as well as mitigating other potential pitfalls in reward learning evaluation.

Given the serious issues with evaluation and training in variable horizon tasks, `imitation` will by default throw an error if training AIRL, GAIL or DRLHP in variable horizon tasks. If you have read this document and understand the problems that variable horizon tasks can cause but still want to train in variable horizon settings, you can override this safety check by setting `allow_variable_horizon=True`. Note this check is not applied for BC or DAgger, which operate on individual transitions (not episodes) and so cannot exploit the information leak.

Usage with `allow_variable_horizon=True` is not officially supported, and we will not optimize `imitation` algorithms to perform well in this situation, as it would not represent real progress. Examples of situations where setting this flag may nonetheless be appropriate include:

1. Investigating the bias introduced by variable horizon tasks – e.g. comparing variable to fixed horizon tasks.
2. For unit tests to verify algorithms continue to run on variable horizon environments.
3. Where the termination condition is trivial (e.g. has the robot fallen over?) and the target behaviour is complex (e.g. solve a Rubik's cube). In this case, while the termination condition still helps reward and imitation learning, the problem remains highly non-trivial even with this information side-channel. However, the existence of this side-channel should of course be prominently disclosed.

See this GitHub issue for further discussion.

### 2.8.1 Non-Support for Infinite Length Horizons

At the moment, we do not support infinite-length horizons. Many of the imitation algorithms, especially those relying on RL, do not easily port over to infinite-horizon setups. Similarly, much of the logging and reward calculation logic assumes the existence of a finite horizon. Although we may explore workarounds in the future, this is not a feature that we can currently support.

## 2.9 Benchmarking `imitation`

The imitation library is benchmarked by running the algorithms BC, DAgger, AIRL and GAIL on five different environments from the seals environment suite each with 10 different random seeds. You will find the benchmark results in the release artifacts, e.g. for the v1.0 release here.

### 2.9.1 Running a Single Benchmark

To run a single benchmark from the commandline, you may use:

```
python -m imitation.scripts.<train_script> <algo> with <algo>_<env>
```

There are two different `train_scripts`: `train_imitation` and `train_adversarial` each running different algorithms:

| train_script | algo |
|---|---|
| train_imitation | bc, dagger |
| train_adversarial | gail, airl |

There are five environment configurations for which we have tuned hyperparameters:

| environment |
|---|
| seals_ant |
| seals_half_cheetah |
| seals_hopper |
| seals_swimmer |
| seals_walker |

If you want to run the same benchmark from a python script, you can use the following code:

```
...
from imitation.scripts.<train_script> import <train_script>_ex
<train_script>_ex.run(command_name="<algo>", named_configs=["<algo>_<env>"])
```

### Inputs

The tuned hyperparameters can be found in `src/imitation/scripts/config/tuned_hps`. For v0.4.0, they correspond to the hyperparameters used in the paper imitation: Clean Imitation Learning Implementations. You may be able to get reasonable performance by using hyperparameters tuned for a similar environment.

The experts and expert demonstrations are loaded from the HuggingFace model hub and are grouped under the HumanCompatibleAI Organization.

### Outputs

The training scripts are sacred experiments which place their output in an output folder structured like this:

```
output
├── airl
│  └── seals-Swimmer-v1
│      └── 20231012_121226_c5c0e4
│          └── sacred -> ../../../sacred/train_adversarial/2
├── dagger
│  └── seals-CartPole-v0
│      └── 20230927_095917_c29dc2
│          └──  sacred -> ../../../sacred/train_imitation/1
└── sacred
    ├── train_adversarial
    │  ├── 1
    │  ├── 2
    │  ├── 3
    │  ├── 4
    │  ├── ...
    │  └── _sources
    └── train_imitation
        ├── 1
        └── _sources
```

In the `sacred` folder all runs are grouped by the training script, and each gets a folder with their run id. That run folder contains

- a `config.json` file with the hyperparameters used for that run
- a `run.json` file with run information with the final score and expert score
- a `cout.txt` file with the stdout of the run

Additionally, there are run folders grouped by algorithm and environment. They contain further log files and model checkpoints as well as a symlink to the corresponding sacred run folder.

Important entries in the json files are:

- `run.json`
  - `command`: The name of the algorithm
  - `result.imit_stats.monitor_return_mean`: the score of a run
  - `result.expert_stats.monitor_return_mean`: the score of the expert policy that was used for a run
- `config.json`
  - `environment.gym_id` The environment name of the run

## 2.9.2 Running the Complete Benchmark Suite

To execute the entire benchmarking suite with 10 seeds for each configuration, you can utilize the `run_all_benchmarks.sh` script. This script will consecutively run all configurations. To optimize the process, consider parallelization options. You can either send all commands to GNU Parallel, use SLURM by invoking `run_all_benchmarks_on_slurm.sh` or split up the lines in multiple scripts to run on multiple machines manually.

### Generating Benchmark Summaries

There are scripts to summarize all runs in a folder in a CSV file or in a markdown file. For the CSV, run:

```
python sacred_output_to_csv.py output/sacred > summary.csv
```

This generates a csv file like this:

```
algo, env, score, expert_score
gail, seals/Walker2d-v1, 2298.883520464286, 2502.8930135576925
gail, seals/Swimmer-v1, 287.33667667857145, 295.40472964423077
airl, seals/Walker2d-v1, 310.4065185178571, 2502.8930135576925
...
```

For a more comprehensive summary that includes aggregate statistics such as mean, standard deviation, IQM (Inter Quartile Mean) with confidence intervals, as recommended by the rliable library, use the following command:

```
python sacred_output_to_markdown_summary output/sacred --output summary.md
```

This will produce a markdown summary file named `summary.md`.

**Hint:** If you have multiple output folders, because you ran different parts of the benchmark on different machines, you can copy the output folders into a common root folder. The above scripts will search all nested directories for folders with a `run.json` and a `config.json` file. For example, calling `python sacred_output_to_csv.py benchmark_runs/ > summary.csv` on an output folder structured like this:

```
benchmark_runs
├── first_batch
│   ├── 1
│   ├── 2
│   ├── 3
```

(continues on next page)

(continued from previous page)

```
│   ├── ...
└── second_batch
    ├── 1
    ├── 2
    ├── 3
    ├── ...
```

will aggregate all runs from both `first_batch` and `second_batch` into a single csv file.

### 2.9.3 Comparing an Algorithm against the Benchmark Runs

If you modified one of the existing algorithms or implemented a new one, you might want to compare it to the benchmark runs to see if there is a significant improvement or not.

If your algorithm has the same file output format as described above, you can use the `compute_probability_of_improvement.py` script to do the comparison. It uses the "Probability of Improvement" metric as recommended by the rliable library.

```
python compute_probability_of_improvement.py <your_runs_dir> <baseline_runs_dir> --
↪baseline-algo <algo>
```

where:

- `your_runs_dir` is the directory containing the runs for your algorithm
- `baseline_runs_dir` is the directory containing runs for a known algorithm. Hint: you do not need to re-run our benchmarks. We provide our run folders as release artifacts.
- `algo` is the algorithm you want to compare against

If `your_runs_dir` contains runs for more than one algorithm, you will have to disambiguate using the `--algo` option.

### 2.9.4 Tuning Hyperparameters

The hyperparameters of any algorithm in imitation can be tuned using `src/imitation/scripts/tuning.py`. The benchmarking hyperparameter configs were generated by tuning the hyperparameters using the search space defined in the `scripts/config/tuning.py`.

The tuning script proceeds in two phases:

1. Tune the hyperparameters using the search space provided.
2. Re-evaluate the best hyperparameter config found in the first phase based on the maximum mean return on a separate set of seeds. Report the mean and standard deviation of these trials.

To use it with the default search space:

```
python -m imitation.scripts.tuning with <algo> 'parallel_run_config.base_named_
↪configs=["<env>"]'
```

In this command:

- `<algo>` provides the default search space and settings for the specific algorithm, which is defined in the `scripts/config/tuning.py`

- `<env>` sets the environment to tune the algorithm in. They are defined in the algo-specifc `scripts/config/train_[adversarial|imitation|preference_comparisons|rl].py` files. For the already tuned environments, use the `<algo>_<env>` named configs here.

See the documentation of `scripts/tuning.py` and `scripts/parallel.py` for many other arguments that can be provided through the command line to change the tuning behavior.

## 2.10 Benchmark Summary

This is a summary of the sacred runs in `benchmark_runs` generated by `sacred_output_to_markdown_summary.py`.

### 2.10.1 Scores

The scores are normalized based on the performance of a random agent as the baseline and the expert as the maximum possible score as explained in this blog post:

```
(score - random_score) / (expert_score - random_score)
```

Aggregate scores and confidence intervals are computed using the rliable library.

#### AIRL

| Environment | Score (mean/std) | Normalized Score (mean/std) | N |
|---|---|---|---|
| seals/Ant-v1 | 2485.889 / 533.471 | 0.981 / 0.184 | 10 |
| seals/HalfCheetah-v1 | 938.450 / 804.871 | 0.627 / 0.412 | 10 |
| seals/Hopper-v1 | 183.780 / 93.295 | 0.921 / 0.373 | 10 |
| seals/Swimmer-v1 | 286.699 / 7.763 | 0.970 / 0.027 | 10 |
| seals/Walker2d-v1 | 1154.921 / 659.564 | 0.461 / 0.264 | 10 |

#### Aggregate Normalized scores

| Metric | Value | 95% CI |
|---|---|---|
| Mean | 0.792 | [0.709, 0.792] |
| IQM | 0.918 | [0.871, 0.974] |

#### BC

| Environment | Score (mean/std) | Normalized Score (mean/std) | N |
|---|---|---|---|
| seals/Ant-v1 | 2090.551 / 180.340 | 0.844 / 0.062 | 10 |
| seals/HalfCheetah-v1 | 1516.476 / 37.487 | 0.923 / 0.019 | 10 |
| seals/Hopper-v1 | 204.271 / 0.609 | 1.003 / 0.002 | 10 |
| seals/Swimmer-v1 | 276.242 / 9.328 | 0.935 / 0.032 | 10 |
| seals/Walker2d-v1 | 2393.254 / 37.641 | 0.956 / 0.015 | 10 |

### Aggregate Normalized scores

| Metric | Value | 95% CI |
|---|---|---|
| Mean | 0.932 | [0.922, 0.932] |
| IQM | 0.941 | [0.941, 0.949] |

### DAGGER

| Environment | Score (mean/std) | Normalized Score (mean/std) | N |
|---|---|---|---|
| seals/Ant-v1 | 2302.527 / 108.315 | 0.957 / 0.052 | 10 |
| seals/HalfCheetah-v1 | 1615.004 / 8.262 | 1.017 / 0.008 | 10 |
| seals/Hopper-v1 | 204.789 / 1.599 | 1.011 / 0.012 | 10 |
| seals/Swimmer-v1 | 283.776 / 6.524 | 0.988 / 0.024 | 10 |
| seals/Walker2d-v1 | 2419.748 / 52.215 | 1.002 / 0.026 | 10 |

### Aggregate Normalized scores

| Metric | Value | 95% CI |
|---|---|---|
| Mean | 0.995 | [0.987, 0.998] |
| IQM | 1.004 | [1.003, 1.008] |

### GAIL

| Environment | Score (mean/std) | Normalized Score (mean/std) | N |
|---|---|---|---|
| seals/Ant-v1 | 2527.566 / 148.034 | 0.995 / 0.051 | 10 |
| seals/HalfCheetah-v1 | 1595.129 / 37.374 | 0.963 / 0.019 | 10 |
| seals/Hopper-v1 | 187.105 / 14.298 | 0.935 / 0.057 | 10 |
| seals/Swimmer-v1 | 249.949 / 74.295 | 0.845 / 0.254 | 10 |
| seals/Walker2d-v1 | 2399.196 / 89.949 | 0.959 / 0.036 | 10 |

### Aggregate Normalized scores

| Metric | Value | 95% CI |
|---|---|---|
| Mean | 0.939 | [0.900, 0.944] |
| IQM | 0.957 | [0.965, 0.970] |

## 2.11 Behavioral Cloning (BC)

Behavioral cloning directly learns a policy by using supervised learning on observation-action pairs from expert demonstrations. It is a simple approach to learning a policy, but the policy often generalizes poorly and does not recover well from errors.

Alternatives to behavioral cloning include *DAgger* (similar but gathers on-policy demonstrations) and *GAIL*/*AIRL* (more robust approaches to learning from demonstrations).

### 2.11.1 Example

Detailed example notebook: *Train an Agent using Behavior Cloning*

```
import numpy as np
import gymnasium as gym
from stable_baselines3.common.evaluation import evaluate_policy

from imitation.algorithms import bc
from imitation.data import rollout
from imitation.data.wrappers import RolloutInfoWrapper
from imitation.policies.serialize import load_policy
from imitation.util.util import make_vec_env

rng = np.random.default_rng(0)
env = make_vec_env(
    "seals:seals/CartPole-v0",
    rng=rng,
    n_envs=1,
    post_wrappers=[lambda env, _: RolloutInfoWrapper(env)],  # for computing rollouts
)
expert = load_policy(
    "ppo-huggingface",
    organization="HumanCompatibleAI",
    env_name="seals-CartPole-v0",
    venv=env,
)
rollouts = rollout.rollout(
    expert,
    env,
    rollout.make_sample_until(min_timesteps=None, min_episodes=50),
    rng=rng,
)
transitions = rollout.flatten_trajectories(rollouts)

bc_trainer = bc.BC(
    observation_space=env.observation_space,
    action_space=env.action_space,
    demonstrations=transitions,
    rng=rng,
)
bc_trainer.train(n_epochs=1)
reward, _ = evaluate_policy(bc_trainer.policy, env, 10)
print("Reward:", reward)
```

### 2.11.2 API

**class** `imitation.algorithms.bc.`**`BC`**`(`**, observation_space, action_space, rng, policy=None, demonstrations=None, batch_size=32, minibatch_size=None, optimizer_cls=<class 'torch.optim.adam.Adam'>, optimizer_kwargs=None, ent_weight=0.001, l2_weight=0.0, device='auto', custom_logger=None*`)`

Bases: *`DemonstrationAlgorithm`*

Behavioral cloning (BC).

Recovers a policy via supervised learning from observation-action pairs.

**`__init__`**`(`**, observation_space, action_space, rng, policy=None, demonstrations=None, batch_size=32, minibatch_size=None, optimizer_cls=<class 'torch.optim.adam.Adam'>, optimizer_kwargs=None, ent_weight=0.001, l2_weight=0.0, device='auto', custom_logger=None*`)`

Builds BC.

**Parameters**

- **`observation_space`** (`Space`) – the observation space of the environment.
- **`action_space`** (`Space`) – the action space of the environment.
- **`rng`** (`Generator`) – the random state to use for the random number generator.
- **`policy`** (`Optional[ActorCriticPolicy]`) – a Stable Baselines3 policy; if unspecified, defaults to *FeedForward32Policy*.
- **`demonstrations`** (`Union[Iterable[`*`Trajectory`*`], Iterable[`*`TransitionMapping`*`], `*`TransitionsMinimal`*`, None]`) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **`batch_size`** (`int`) – The number of samples in each batch of expert data.
- **`minibatch_size`** (`Optional[int]`) – size of minibatch to calculate gradients over. The gradients are accumulated until *batch_size* examples are processed before making an optimization step. This is useful in GPU training to reduce memory usage, since fewer examples are loaded into memory at once, facilitating training with larger batch sizes, but is generally slower. Must be a factor of *batch_size*. Optional, defaults to *batch_size*.
- **`optimizer_cls`** (`Type[Optimizer]`) – optimiser to use for supervised training.
- **`optimizer_kwargs`** (`Optional[Mapping[str, Any]]`) – keyword arguments, excluding learning rate and weight decay, for optimiser construction.
- **`ent_weight`** (`float`) – scaling applied to the policy's entropy regularization.
- **`l2_weight`** (`float`) – scaling applied to the policy's L2 regularization.
- **`device`** (`Union[str, device]`) – name/identity of device to place policy on.
- **`custom_logger`** (`Optional[`*`HierarchicalLogger`*`]`) – Where to log to; if None (default), creates a new logger.

**Raises**

**`ValueError`** – If *weight_decay* is specified in *optimizer_kwargs* (use the parameter *l2_weight* instead), or if the batch size is not a multiple of the minibatch size.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**property policy: ActorCriticPolicy**

Returns a policy imitating the demonstration data.

**Return type**

`ActorCriticPolicy`

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (`Union[Iterable[`*`Trajectory`*`], Iterable[`*`TransitionMapping`*`], `*`TransitionsMinimal`*`]`) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**

`None`

**train**(**, n_epochs=None, n_batches=None, on_epoch_end=None, on_batch_end=None, log_interval=500, log_rollouts_venv=None, log_rollouts_n_episodes=5, progress_bar=True, reset_tensorboard=False*)

Train with supervised learning for some number of epochs.

Here an 'epoch' is just a complete pass through the expert data loader, as set by *self.set_expert_data_loader()*. Note, that when you specify *n_batches* smaller than the number of batches in an epoch, the *on_epoch_end* callback will never be called.

**Parameters**

- **n_epochs** (`Optional[int]`) – Number of complete passes made through expert data before ending training. Provide exactly one of *n_epochs* and *n_batches*.
- **n_batches** (`Optional[int]`) – Number of batches loaded from dataset before ending training. Provide exactly one of *n_epochs* and *n_batches*.
- **on_epoch_end** (`Optional[Callable[[], None]]`) – Optional callback with no parameters to run at the end of each epoch.
- **on_batch_end** (`Optional[Callable[[], None]]`) – Optional callback with no parameters to run at the end of each batch.
- **log_interval** (`int`) – Log stats after every log_interval batches.
- **log_rollouts_venv** (`Optional[VecEnv]`) – If not None, then this VecEnv (whose observation and actions spaces must match *self.observation_space* and *self.action_space*) is used to generate rollout stats, including average return and average episode length. If None, then no rollouts are generated.
- **log_rollouts_n_episodes** (`int`) – Number of rollouts to generate when calculating rollout stats. Non-positive number disables rollouts.
- **progress_bar** (`bool`) – If True, then show a progress bar during training.
- **reset_tensorboard** (`bool`) – If True, then start plotting to Tensorboard from x=0 even if *.train()* logged to Tensorboard previously. Has no practical effect if *.train()* is being called for the first time.

## 2.12 Generative Adversarial Imitation Learning (GAIL)

GAIL learns a policy by simultaneously training it with a discriminator that aims to distinguish expert trajectories against trajectories from the learned policy.

**Note:** GAIL paper: Generative Adversarial Imitation Learning

### 2.12.1 Example

Detailed example notebook: *Train an Agent using Generative Adversarial Imitation Learning*

```python
import numpy as np
import gymnasium as gym
from stable_baselines3 import PPO
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3.ppo import MlpPolicy

from imitation.algorithms.adversarial.gail import GAIL
from imitation.data import rollout
from imitation.data.wrappers import RolloutInfoWrapper
from imitation.policies.serialize import load_policy
from imitation.rewards.reward_nets import BasicRewardNet
from imitation.util.networks import RunningNorm
from imitation.util.util import make_vec_env

SEED = 42

env = make_vec_env(
    "seals:seals/CartPole-v0",
    rng=np.random.default_rng(SEED),
    n_envs=8,
    post_wrappers=[lambda env, _: RolloutInfoWrapper(env)],  # to compute rollouts
)
expert = load_policy(
    "ppo-huggingface",
    organization="HumanCompatibleAI",
    env_name="seals-CartPole-v0",
    venv=env,
)

rollouts = rollout.rollout(
    expert,
    env,
    rollout.make_sample_until(min_timesteps=None, min_episodes=60),
    rng=np.random.default_rng(SEED),
)

learner = PPO(
    env=env,
    policy=MlpPolicy,
    batch_size=64,
    ent_coef=0.0,
    learning_rate=0.0004,
    gamma=0.95,
```

(continues on next page)

(continued from previous page)

```
    n_epochs=5,
    seed=SEED,
)
reward_net = BasicRewardNet(
    observation_space=env.observation_space,
    action_space=env.action_space,
    normalize_input_layer=RunningNorm,
)
gail_trainer = GAIL(
    demonstrations=rollouts,
    demo_batch_size=1024,
    gen_replay_buffer_capacity=512,
    n_disc_updates_per_round=8,
    venv=env,
    gen_algo=learner,
    reward_net=reward_net,
)

# evaluate the learner before training
env.seed(SEED)
learner_rewards_before_training, _ = evaluate_policy(
    learner, env, 100, return_episode_rewards=True,
)

# train the learner and evaluate again
gail_trainer.train(20000)  # Train for 800_000 steps to match expert.
env.seed(SEED)
learner_rewards_after_training, _ = evaluate_policy(
    learner, env, 100, return_episode_rewards=True,
)

print("mean reward after training:", np.mean(learner_rewards_after_training))
print("mean reward before training:", np.mean(learner_rewards_before_training))
```

### 2.12.2 API

**class** imitation.algorithms.adversarial.gail.**GAIL**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, **kwargs*)

Bases: *AdversarialTrainer*

Generative Adversarial Imitation Learning (GAIL).

**__init__**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, **kwargs*)

Generative Adversarial Imitation Learning.

**Parameters**

- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **demo_batch_size** (int) – The number of samples in each batch of expert data. The discriminator batch size is twice this number because each discriminator batch contains a generator sample for every expert sample.

- **venv** (VecEnv) – The vectorized environment to train in.
- **gen_algo** (BaseAlgorithm) – The generator RL algorithm that is trained to maximize discriminator confusion. Environment and logger will be set to *venv* and *custom_logger*.
- **reward_net** (*RewardNet*) – a Torch module that takes an observation, action and next observation tensor as input, then computes the logits. Used as the GAIL discriminator.
- ****kwargs** – Passed through to *AdversarialTrainer.__init__*.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**property logger: *HierarchicalLogger***

**Return type**
*HierarchicalLogger*

**logits_expert_is_high**(*state*, *action*, *next_state*, *done*, *log_policy_act_prob=None*)

Compute the discriminator's logits for each state-action sample.

**Parameters**

- **state** (Tensor) – The state of the environment at the time of the action.
- **action** (Tensor) – The action taken by the expert or generator.
- **next_state** (Tensor) – The state of the environment after the action.
- **done** (Tensor) – whether a *terminal state* (as defined under the MDP of the task) has been reached.
- **log_policy_act_prob** (Optional[Tensor]) – The log probability of the action taken by the generator, log $P(a|s)$.

**Return type**
Tensor

**Returns**
The logits of the discriminator for each state-action sample.

**property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**
BasePolicy

**property reward_test: *RewardNet***

Reward used to train policy at "test" time after adversarial training.

**Return type**
*RewardNet*

**property reward_train: *RewardNet***

Reward used to train generator policy.

**Return type**
*RewardNet*

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**

None

**train**(*total_timesteps*, *callback=None*)

Alternates between training the generator and discriminator.

Every "round" consists of a call to *train_gen(self.gen_train_timesteps)*, a call to *train_disc*, and finally a call to *callback(round)*.

Training ends once an additional "round" would cause the number of transitions sampled from the environment to exceed *total_timesteps*.

**Parameters**

- **total_timesteps** (int) – An upper bound on the number of transitions to sample from the environment during training.
- **callback** (Optional[Callable[[int], None]]) – A function called at the end of every round which takes in a single argument, the round number. Round numbers are in *range(total_timesteps // self.gen_train_timesteps)*.

**Return type**

None

**train_disc**(*, *expert_samples=None*, *gen_samples=None*)

Perform a single discriminator update, optionally using provided samples.

**Parameters**

- **expert_samples** (Optional[Mapping]) – Transition samples from the expert in dictionary form. If provided, must contain keys corresponding to every field of the *Transitions* dataclass except "infos". All corresponding values can be either NumPy arrays or Tensors. Extra keys are ignored. Must contain *self.demo_batch_size* samples. If this argument is not provided, then *self.demo_batch_size* expert samples from *self.demo_data_loader* are used by default.
- **gen_samples** (Optional[Mapping]) – Transition samples from the generator policy in same dictionary form as *expert_samples*. If provided, must contain exactly *self.demo_batch_size* samples. If not provided, then take *len(expert_samples)* samples from the generator replay buffer.

**Return type**

Mapping[str, float]

**Returns**

Statistics for discriminator (e.g. loss, accuracy).

**train_gen**(*total_timesteps=None*, *learn_kwargs=None*)

Trains the generator to maximize the discriminator loss.

After the end of training populates the generator replay buffer (used in discriminator training) with *self.disc_batch_size* transitions.

**Parameters**

- **total_timesteps** (Optional[int]) – The number of transitions to sample from *self.venv_train* during training. By default, *self.gen_train_timesteps*.
- **learn_kwargs** (Optional[Mapping]) – kwargs for the Stable Baselines *RLModel.learn()* method.

**Return type**
None

**venv: VecEnv**

The original vectorized environment.

**venv_train: VecEnv**

Like *self.venv*, but wrapped with train reward unless in debug mode.

If *debug_use_ground_truth=True* was passed into the initializer then *self.venv_train* is the same as *self.venv*.

**venv_wrapped: VecEnvWrapper**

**class** imitation.algorithms.adversarial.common.**AdversarialTrainer**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, demo_minibatch_size=None, n_disc_updates_per_round=2, log_dir='output/', disc_opt_cls=<class 'torch.optim.adam.Adam'>, disc_opt_kwargs=None, gen_train_timesteps=None, gen_replay_buffer_capacity=None, custom_logger=None, init_tensorboard=False, init_tensorboard_graph=False, debug_use_ground_truth=False, allow_variable_horizon=False*)

Bases: *DemonstrationAlgorithm*[*Transitions*]

Base class for adversarial imitation learning algorithms like GAIL and AIRL.

**__init__**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, demo_minibatch_size=None, n_disc_updates_per_round=2, log_dir='output/', disc_opt_cls=<class 'torch.optim.adam.Adam'>, disc_opt_kwargs=None, gen_train_timesteps=None, gen_replay_buffer_capacity=None, custom_logger=None, init_tensorboard=False, init_tensorboard_graph=False, debug_use_ground_truth=False, allow_variable_horizon=False*)

Builds AdversarialTrainer.

**Parameters**

- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **demo_batch_size** (int) – The number of samples in each batch of expert data. The discriminator batch size is twice this number because each discriminator batch contains a generator sample for every expert sample.
- **venv** (VecEnv) – The vectorized environment to train in.
- **gen_algo** (BaseAlgorithm) – The generator RL algorithm that is trained to maximize discriminator confusion. Environment and logger will be set to *venv* and *custom_logger*.
- **reward_net** (*RewardNet*) – a Torch module that takes an observation, action and next observation tensors as input and computes a reward signal.
- **demo_minibatch_size** (Optional[int]) – size of minibatch to calculate gradients over. The gradients are accumulated until the entire batch is processed before making an optimization step. This is useful in GPU training to reduce memory usage, since fewer examples are loaded into memory at once, facilitating training with larger batch sizes, but is generally slower. Must be a factor of *demo_batch_size*. Optional, defaults to *demo_batch_size*.
- **n_disc_updates_per_round** (int) – The number of discriminator updates after each round of generator updates in AdversarialTrainer.learn().
- **log_dir** (Union[str, bytes, PathLike]) – Directory to store TensorBoard logs, plots, etc. in.
- **disc_opt_cls** (Type[Optimizer]) – The optimizer for discriminator training.
- **disc_opt_kwargs** (Optional[Mapping]) – Parameters for discriminator training.
- **gen_train_timesteps** (Optional[int]) – The number of steps to train the generator policy for each iteration. If None, then defaults to the batch size (for on-policy) or number of environments (for off-policy).
- **gen_replay_buffer_capacity** (Optional[int]) – The capacity of the generator replay buffer (the number of obs-action-obs samples from the generator that can be stored). By default this is equal to *gen_train_timesteps*, meaning that we sample only from the most recent batch of generator samples.
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **init_tensorboard** (bool) – If True, makes various discriminator TensorBoard summaries.
- **init_tensorboard_graph** (bool) – If both this and *init_tensorboard* are True, then write a Tensorboard graph summary to disk.
- **debug_use_ground_truth** (bool) – If True, use the ground truth reward for *self.train_env*. This disables the reward wrapping that would normally replace the environment reward with the learned reward. This is useful for sanity checking that the policy training is functional.
- **allow_variable_horizon** (bool) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety

check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/guide/variable_horizon.html before overriding this.

**Raises**

`ValueError` – if the batch size is not a multiple of the minibatch size.

**`allow_variable_horizon: bool`**

If True, allow variable horizon trajectories; otherwise error if detected.

**`property logger: HierarchicalLogger`**

**Return type**

`HierarchicalLogger`

**`abstract logits_expert_is_high`**(*state*, *action*, *next_state*, *done*, *log_policy_act_prob=None*)

Compute the discriminator's logits for each state-action sample.

A high value corresponds to predicting expert, and a low value corresponds to predicting generator.

**Parameters**

- **`state`** (`Tensor`) – state at time t, of shape *(batch_size,)* + *state_shape*.
- **`action`** (`Tensor`) – action taken at time t, of shape *(batch_size,)* + *action_shape*.
- **`next_state`** (`Tensor`) – state at time t+1, of shape *(batch_size,)* + *state_shape*.
- **`done`** (`Tensor`) – binary episode completion flag after action at time t, of shape *(batch_size,)*.
- **`log_policy_act_prob`** (`Optional[Tensor]`) – log probability of generator policy taking *action* at time t.

**Return type**

`Tensor`

**Returns**

Discriminator logits of shape *(batch_size,)*. A high output indicates an expert-like transition.

**`property policy: BasePolicy`**

Returns a policy imitating the demonstration data.

**Return type**

`BasePolicy`

**`abstract property reward_test: RewardNet`**

Reward used to train policy at "test" time after adversarial training.

**Return type**

`RewardNet`

**`abstract property reward_train: RewardNet`**

Reward used to train generator policy.

**Return type**

`RewardNet`

**`set_demonstrations`**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**

None

**train**(*total_timesteps*, *callback=None*)

Alternates between training the generator and discriminator.

Every "round" consists of a call to *train_gen(self.gen_train_timesteps)*, a call to *train_disc*, and finally a call to *callback(round)*.

Training ends once an additional "round" would cause the number of transitions sampled from the environment to exceed *total_timesteps*.

**Parameters**

- **total_timesteps** (int) – An upper bound on the number of transitions to sample from the environment during training.
- **callback** (Optional[Callable[[int], None]]) – A function called at the end of every round which takes in a single argument, the round number. Round numbers are in *range(total_timesteps // self.gen_train_timesteps)*.

**Return type**

None

**train_disc**(*, *expert_samples=None*, *gen_samples=None*)

Perform a single discriminator update, optionally using provided samples.

**Parameters**

- **expert_samples** (Optional[Mapping]) – Transition samples from the expert in dictionary form. If provided, must contain keys corresponding to every field of the *Transitions* dataclass except "infos". All corresponding values can be either NumPy arrays or Tensors. Extra keys are ignored. Must contain *self.demo_batch_size* samples. If this argument is not provided, then *self.demo_batch_size* expert samples from *self.demo_data_loader* are used by default.
- **gen_samples** (Optional[Mapping]) – Transition samples from the generator policy in same dictionary form as *expert_samples*. If provided, must contain exactly *self.demo_batch_size* samples. If not provided, then take *len(expert_samples)* samples from the generator replay buffer.

**Return type**

Mapping[str, float]

**Returns**

Statistics for discriminator (e.g. loss, accuracy).

**train_gen**(*total_timesteps=None*, *learn_kwargs=None*)

Trains the generator to maximize the discriminator loss.

After the end of training populates the generator replay buffer (used in discriminator training) with *self.disc_batch_size* transitions.

**Parameters**

- **total_timesteps** (Optional[int]) – The number of transitions to sample from *self.venv_train* during training. By default, *self.gen_train_timesteps*.
- **learn_kwargs** (Optional[Mapping]) – kwargs for the Stable Baselines *RLModel.learn()* method.

**Return type**

None

**venv: VecEnv**

The original vectorized environment.

**venv_train: VecEnv**

Like *self.venv*, but wrapped with train reward unless in debug mode.

If *debug_use_ground_truth=True* was passed into the initializer then *self.venv_train* is the same as *self.venv*.

**venv_wrapped: VecEnvWrapper**

## 2.13 Adversarial Inverse Reinforcement Learning (AIRL)

AIRL, similar to *GAIL*, adversarially trains a policy against a discriminator that aims to distinguish the expert demonstrations from the learned policy. Unlike GAIL, AIRL recovers a reward function that is more generalizable to changes in environment dynamics.

The expert policy must be stochastic.

**Note:** AIRL paper: Learning Robust Rewards with Adversarial Inverse Reinforcement Learning

### 2.13.1 Example

Detailed example notebook: *Train an Agent using Adversarial Inverse Reinforcement Learning*

```
import numpy as np
import gymnasium as gym
from stable_baselines3 import PPO
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3.ppo import MlpPolicy

from imitation.algorithms.adversarial.airl import AIRL
from imitation.data import rollout
from imitation.data.wrappers import RolloutInfoWrapper
from imitation.policies.serialize import load_policy
from imitation.rewards.reward_nets import BasicShapedRewardNet
from imitation.util.networks import RunningNorm
from imitation.util.util import make_vec_env

SEED = 42

env = make_vec_env(
    "seals:seals/CartPole-v0",
    rng=np.random.default_rng(SEED),
    n_envs=8,
    post_wrappers=[lambda env, _: RolloutInfoWrapper(env)],  # to compute rollouts
```

(continues on next page)

(continued from previous page)

```python
)
expert = load_policy(
    "ppo-huggingface",
    organization="HumanCompatibleAI",
    env_name="seals-CartPole-v0",
    venv=env,
)
rollouts = rollout.rollout(
    expert,
    env,
    rollout.make_sample_until(min_episodes=60),
    rng=np.random.default_rng(SEED),
)

learner = PPO(
    env=env,
    policy=MlpPolicy,
    batch_size=64,
    ent_coef=0.0,
    learning_rate=0.0005,
    gamma=0.95,
    clip_range=0.1,
    vf_coef=0.1,
    n_epochs=5,
    seed=SEED,
)
reward_net = BasicShapedRewardNet(
    observation_space=env.observation_space,
    action_space=env.action_space,
    normalize_input_layer=RunningNorm,
)
airl_trainer = AIRL(
    demonstrations=rollouts,
    demo_batch_size=2048,
    gen_replay_buffer_capacity=512,
    n_disc_updates_per_round=16,
    venv=env,
    gen_algo=learner,
    reward_net=reward_net,
)

env.seed(SEED)
learner_rewards_before_training, _ = evaluate_policy(
    learner, env, 100, return_episode_rewards=True,
)
airl_trainer.train(20000)  # Train for 2_000_000 steps to match expert.
env.seed(SEED)
learner_rewards_after_training, _ = evaluate_policy(
    learner, env, 100, return_episode_rewards=True,
)

print("mean reward after training:", np.mean(learner_rewards_after_training))
print("mean reward before training:", np.mean(learner_rewards_before_training))
```

### 2.13.2 API

**class** imitation.algorithms.adversarial.airl.**AIRL**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, **kwargs*)

Bases: *AdversarialTrainer*

Adversarial Inverse Reinforcement Learning (AIRL).

**__init__**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, **kwargs*)

Builds an AIRL trainer.

**Parameters**

- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **demo_batch_size** (int) – The number of samples in each batch of expert data. The discriminator batch size is twice this number because each discriminator batch contains a generator sample for every expert sample.
- **venv** (VecEnv) – The vectorized environment to train in.
- **gen_algo** (BaseAlgorithm) – The generator RL algorithm that is trained to maximize discriminator confusion. Environment and logger will be set to *venv* and *custom_logger*.
- **reward_net** (*RewardNet*) – Reward network; used as part of AIRL discriminator.
- ****kwargs** – Passed through to *AdversarialTrainer.__init__*.

**Raises**

**TypeError** – If *gen_algo.policy* does not have an *evaluate_actions* attribute (present in *ActorCriticPolicy*), needed to compute log-probability of actions.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**property logger: *HierarchicalLogger***

**Return type**

*HierarchicalLogger*

**logits_expert_is_high**(*state, action, next_state, done, log_policy_act_prob=None*)

Compute the discriminator's logits for each state-action sample.

In Fu's AIRL paper (https://arxiv.org/pdf/1710.11248.pdf), the discriminator output was given as

$$D_\theta(s,a) = \frac{\exp r_\theta(s,a)}{\exp r_\theta(s,a) + \pi(a|s)}$$

with a high value corresponding to the expert and a low value corresponding to the generator.

In other words, the discriminator output is the probability that the action is taken by the expert rather than the generator.

The logit of the above is given as

$$\operatorname{logit}(D_\theta(s,a)) = r_\theta(s,a) - \log \pi(a|s)$$

which is what is returned by this function.

**Parameters**

- **state** (Tensor) – The state of the environment at the time of the action.
- **action** (Tensor) – The action taken by the expert or generator.
- **next_state** (Tensor) – The state of the environment after the action.
- **done** (Tensor) – whether a *terminal state* (as defined under the MDP of the task) has been reached.
- **log_policy_act_prob** (Optional[Tensor]) – The log probability of the action taken by the generator, $\log \pi(a|s)$.

**Return type**

Tensor

**Returns**

The logits of the discriminator for each state-action sample.

**Raises**

**TypeError** – If *log_policy_act_prob* is None.

**property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**

BasePolicy

**property reward_test: *RewardNet***

Returns the unshaped version of reward network used for testing.

**Return type**

*RewardNet*

**property reward_train: *RewardNet***

Reward used to train generator policy.

**Return type**

*RewardNet*

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**

None

**train**(*total_timesteps*, *callback=None*)

Alternates between training the generator and discriminator.

Every "round" consists of a call to *train_gen(self.gen_train_timesteps)*, a call to *train_disc*, and finally a call to *callback(round)*.

Training ends once an additional "round" would cause the number of transitions sampled from the environment to exceed *total_timesteps*.

**Parameters**

- **total_timesteps** (int) – An upper bound on the number of transitions to sample from the environment during training.
- **callback** (Optional[Callable[[int], None]]) – A function called at the end of every round which takes in a single argument, the round number. Round numbers are in *range(total_timesteps // self.gen_train_timesteps)*.

**Return type**
None

**train_disc**(***, *expert_samples=None*, *gen_samples=None*)

Perform a single discriminator update, optionally using provided samples.

**Parameters**

- **expert_samples** (Optional[Mapping]) – Transition samples from the expert in dictionary form. If provided, must contain keys corresponding to every field of the *Transitions* dataclass except "infos". All corresponding values can be either NumPy arrays or Tensors. Extra keys are ignored. Must contain *self.demo_batch_size* samples. If this argument is not provided, then *self.demo_batch_size* expert samples from *self.demo_data_loader* are used by default.
- **gen_samples** (Optional[Mapping]) – Transition samples from the generator policy in same dictionary form as *expert_samples*. If provided, must contain exactly *self.demo_batch_size* samples. If not provided, then take *len(expert_samples)* samples from the generator replay buffer.

**Return type**
Mapping[str, float]

**Returns**
Statistics for discriminator (e.g. loss, accuracy).

**train_gen**(*total_timesteps=None*, *learn_kwargs=None*)

Trains the generator to maximize the discriminator loss.

After the end of training populates the generator replay buffer (used in discriminator training) with *self.disc_batch_size* transitions.

**Parameters**

- **total_timesteps** (Optional[int]) – The number of transitions to sample from *self.venv_train* during training. By default, *self.gen_train_timesteps*.
- **learn_kwargs** (Optional[Mapping]) – kwargs for the Stable Baselines *RLModel.learn()* method.

**Return type**
None

**venv: VecEnv**

The original vectorized environment.

**venv_train: VecEnv**

Like *self.venv*, but wrapped with train reward unless in debug mode.

If *debug_use_ground_truth=True* was passed into the initializer then *self.venv_train* is the same as *self.venv*.

**venv_wrapped: VecEnvWrapper**

**class** imitation.algorithms.adversarial.common.**AdversarialTrainer**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, demo_minibatch_size=None, n_disc_updates_per_round=2, log_dir='output/', disc_opt_cls=<class 'torch.optim.adam.Adam'>, disc_opt_kwargs=None, gen_train_timesteps=None, gen_replay_buffer_capacity=None, custom_logger=None, init_tensorboard=False, init_tensorboard_graph=False, debug_use_ground_truth=False, allow_variable_horizon=False*)

Bases: *DemonstrationAlgorithm*[*Transitions*]

Base class for adversarial imitation learning algorithms like GAIL and AIRL.

**__init__**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, demo_minibatch_size=None, n_disc_updates_per_round=2, log_dir='output/', disc_opt_cls=<class 'torch.optim.adam.Adam'>, disc_opt_kwargs=None, gen_train_timesteps=None, gen_replay_buffer_capacity=None, custom_logger=None, init_tensorboard=False, init_tensorboard_graph=False, debug_use_ground_truth=False, allow_variable_horizon=False*)

Builds AdversarialTrainer.

**Parameters**

- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **demo_batch_size** (int) – The number of samples in each batch of expert data. The discriminator batch size is twice this number because each discriminator batch contains a generator sample for every expert sample.
- **venv** (VecEnv) – The vectorized environment to train in.
- **gen_algo** (BaseAlgorithm) – The generator RL algorithm that is trained to maximize discriminator confusion. Environment and logger will be set to *venv* and *custom_logger*.
- **reward_net** (*RewardNet*) – a Torch module that takes an observation, action and next

observation tensors as input and computes a reward signal.

- **demo_minibatch_size** (`Optional[int]`) – size of minibatch to calculate gradients over. The gradients are accumulated until the entire batch is processed before making an optimization step. This is useful in GPU training to reduce memory usage, since fewer examples are loaded into memory at once, facilitating training with larger batch sizes, but is generally slower. Must be a factor of *demo_batch_size*. Optional, defaults to *demo_batch_size*.
- **n_disc_updates_per_round** (`int`) – The number of discriminator updates after each round of generator updates in AdversarialTrainer.learn().
- **log_dir** (`Union[str, bytes, PathLike]`) – Directory to store TensorBoard logs, plots, etc. in.
- **disc_opt_cls** (`Type[Optimizer]`) – The optimizer for discriminator training.
- **disc_opt_kwargs** (`Optional[Mapping]`) – Parameters for discriminator training.
- **gen_train_timesteps** (`Optional[int]`) – The number of steps to train the generator policy for each iteration. If None, then defaults to the batch size (for on-policy) or number of environments (for off-policy).
- **gen_replay_buffer_capacity** (`Optional[int]`) – The capacity of the generator replay buffer (the number of obs-action-obs samples from the generator that can be stored). By default this is equal to *gen_train_timesteps*, meaning that we sample only from the most recent batch of generator samples.
- **custom_logger** (`Optional[HierarchicalLogger]`) – Where to log to; if None (default), creates a new logger.
- **init_tensorboard** (`bool`) – If True, makes various discriminator TensorBoard summaries.
- **init_tensorboard_graph** (`bool`) – If both this and *init_tensorboard* are True, then write a Tensorboard graph summary to disk.
- **debug_use_ground_truth** (`bool`) – If True, use the ground truth reward for *self.train_env*. This disables the reward wrapping that would normally replace the environment reward with the learned reward. This is useful for sanity checking that the policy training is functional.
- **allow_variable_horizon** (`bool`) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/guide/variable_horizon.html before overriding this.

**Raises**

**ValueError** – if the batch size is not a multiple of the minibatch size.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**abstract logits_expert_is_high** (*state*, *action*, *next_state*, *done*, *log_policy_act_prob=None*)

Compute the discriminator's logits for each state-action sample.

A high value corresponds to predicting expert, and a low value corresponds to predicting generator.

**Parameters**

- **state** (`Tensor`) – state at time t, of shape *(batch_size,)* + *state_shape*.
- **action** (`Tensor`) – action taken at time t, of shape *(batch_size,)* + *action_shape*.

- **next_state** (`Tensor`) – state at time t+1, of shape *(batch_size,)* + *state_shape*.
- **done** (`Tensor`) – binary episode completion flag after action at time t, of shape *(batch_size,)*.
- **log_policy_act_prob** (`Optional[Tensor]`) – log probability of generator policy taking *action* at time t.

**Return type**
`Tensor`

**Returns**
Discriminator logits of shape *(batch_size,)*. A high output indicates an expert-like transition.

**property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**
`BasePolicy`

**abstract property reward_test: *RewardNet***

Reward used to train policy at "test" time after adversarial training.

**Return type**
*`RewardNet`*

**abstract property reward_train: *RewardNet***

Reward used to train generator policy.

**Return type**
*`RewardNet`*

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**
**demonstrations** (`Union[Iterable[`*`Trajectory`*`], Iterable[`*`TransitionMapping`*`], `*`TransitionsMinimal`*`]`) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**
`None`

**train**(*total_timesteps*, *callback=None*)

Alternates between training the generator and discriminator.

Every "round" consists of a call to *train_gen(self.gen_train_timesteps)*, a call to *train_disc*, and finally a call to *callback(round)*.

Training ends once an additional "round" would cause the number of transitions sampled from the environment to exceed *total_timesteps*.

**Parameters**

- **total_timesteps** (`int`) – An upper bound on the number of transitions to sample from the environment during training.
- **callback** (`Optional[Callable[[int], None]]`) – A function called at the end of every round which takes in a single argument, the round number. Round numbers are in *range(total_timesteps // self.gen_train_timesteps)*.

**Return type**
None

**train_disc**(*, *expert_samples=None*, *gen_samples=None*)

Perform a single discriminator update, optionally using provided samples.

**Parameters**

- **expert_samples** (Optional[Mapping]) – Transition samples from the expert in dictionary form. If provided, must contain keys corresponding to every field of the *Transitions* dataclass except "infos". All corresponding values can be either NumPy arrays or Tensors. Extra keys are ignored. Must contain *self.demo_batch_size* samples. If this argument is not provided, then *self.demo_batch_size* expert samples from *self.demo_data_loader* are used by default.
- **gen_samples** (Optional[Mapping]) – Transition samples from the generator policy in same dictionary form as *expert_samples*. If provided, must contain exactly *self.demo_batch_size* samples. If not provided, then take *len(expert_samples)* samples from the generator replay buffer.

**Return type**
Mapping[str, float]

**Returns**
Statistics for discriminator (e.g. loss, accuracy).

**train_gen**(*total_timesteps=None*, *learn_kwargs=None*)

Trains the generator to maximize the discriminator loss.

After the end of training populates the generator replay buffer (used in discriminator training) with *self.disc_batch_size* transitions.

**Parameters**

- **total_timesteps** (Optional[int]) – The number of transitions to sample from *self.venv_train* during training. By default, *self.gen_train_timesteps*.
- **learn_kwargs** (Optional[Mapping]) – kwargs for the Stable Baselines *RLModel.learn()* method.

**Return type**
None

**venv: VecEnv**

The original vectorized environment.

**venv_train: VecEnv**

Like *self.venv*, but wrapped with train reward unless in debug mode.

If *debug_use_ground_truth=True* was passed into the initializer then *self.venv_train* is the same as *self.venv*.

**venv_wrapped: VecEnvWrapper**

## 2.14 DAgger

DAgger (Dataset Aggregation) iteratively trains a policy using supervised learning on a dataset of observation-action pairs from expert demonstrations (like *behavioral cloning*), runs the policy to gather observations, queries the expert for good actions on those observations, and adds the newly labeled observations to the dataset. DAgger improves on behavioral cloning by training on a dataset that better resembles the observations the trained policy is likely to encounter, but it requires querying the expert online.

**Note:** DAgger paper: A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning

### 2.14.1 Example

Detailed example notebook: *Train an Agent using the DAgger Algorithm*

```
import tempfile

import numpy as np
import gymnasium as gym
from stable_baselines3.common.evaluation import evaluate_policy

from imitation.algorithms import bc
from imitation.algorithms.dagger import SimpleDAggerTrainer
from imitation.policies.serialize import load_policy
from imitation.util.util import make_vec_env

rng = np.random.default_rng(0)
env = make_vec_env(
    "seals:seals/CartPole-v0",
    rng=rng,
)
expert = load_policy(
    "ppo-huggingface",
    organization="HumanCompatibleAI",
    env_name="seals-CartPole-v0",
    venv=env,
)

bc_trainer = bc.BC(
    observation_space=env.observation_space,
    action_space=env.action_space,
    rng=rng,
)
with tempfile.TemporaryDirectory(prefix="dagger_example_") as tmpdir:
    print(tmpdir)
    dagger_trainer = SimpleDAggerTrainer(
        venv=env,
        scratch_dir=tmpdir,
        expert_policy=expert,
        bc_trainer=bc_trainer,
        rng=rng,
    )
    dagger_trainer.train(8_000)
```

(continues on next page)

(continued from previous page)

```
reward, _ = evaluate_policy(dagger_trainer.policy, env, 10)
print("Reward:", reward)
```

### 2.14.2 API

**class** imitation.algorithms.dagger.**InteractiveTrajectoryCollector**(*venv*, *get_robot_acts*, *beta*, *save_dir*, *rng*)

Bases: `VecEnvWrapper`

DAgger VecEnvWrapper for querying and saving expert actions.

Every call to *.step(actions)* accepts and saves expert actions to *self.save_dir*, but only forwards expert actions to the wrapped VecEnv with probability *self.beta*. With probability *1 - self.beta*, a "robot" action (i.e an action from the imitation policy) is forwarded instead.

Demonstrations are saved as *TrajectoryWithRew* to *self.save_dir* at the end of every episode.

**__init__**(*venv*, *get_robot_acts*, *beta*, *save_dir*, *rng*)

Builds InteractiveTrajectoryCollector.

**Parameters**

- **venv** (`VecEnv`) – vectorized environment to sample trajectories from.
- **get_robot_acts** (`Callable[[ndarray], ndarray]`) – get robot actions that can be substituted for human actions. Takes a vector of observations as input & returns a vector of actions.
- **beta** (`float`) – fraction of the time to use action given to .step() instead of robot action. The choice of robot or human action is independently randomized for each individual *Env* at every timestep.
- **save_dir** (`Union[str, bytes, PathLike]`) – directory to save collected trajectories in.
- **rng** (`Generator`) – random state for random number generation.

**close**()

Clean up the environment's resources.

**Return type**

`None`

**env_is_wrapped**(*wrapper_class*, *indices=None*)

Check if environments are wrapped with a given wrapper.

**Parameters**

- **method_name** – The name of the environment method to invoke.
- **indices** (`Union[None, int, Iterable[int]]`) – Indices of envs whose method to call
- **method_args** – Any positional arguments to provide in the call
- **method_kwargs** – Any keyword arguments to provide in the call

**Return type**

`List[bool]`

**Returns**
True if the env is wrapped, False otherwise, for each env queried.

**`env_method`**(*method_name*, **method_args*, *indices=None*, ***method_kwargs*)

Call instance methods of vectorized environments.

**Parameters**

- **`method_name`** (`str`) – The name of the environment method to invoke.
- **`indices`** (`Union[None, int, Iterable[int]]`) – Indices of envs whose method to call
- **`method_args`** – Any positional arguments to provide in the call
- **`method_kwargs`** – Any keyword arguments to provide in the call

**Return type**
`List[Any]`

**Returns**
List of items returned by the environment's method call

**`get_attr`**(*attr_name*, *indices=None*)

Return attribute from vectorized environment.

**Parameters**

- **`attr_name`** (`str`) – The name of the attribute whose value to return
- **`indices`** (`Union[None, int, Iterable[int]]`) – Indices of envs to get attribute from

**Return type**
`List[Any]`

**Returns**
List of values of 'attr_name' in all environments

**`get_images`**()

Return RGB images from each environment when available

**Return type**
`Sequence[Optional[ndarray]]`

**`getattr_depth_check`**(*name*, *already_found*)

See base class.

**Return type**
`Optional[str]`

**Returns**
name of module whose attribute is being shadowed, if any.

**`getattr_recursive`**(*name*)

Recursively check wrappers to find attribute.

**Parameters**
**`name`** (`str`) – name of attribute to look for

**Return type**
`Any`

**Returns**
attribute

**render**(*mode=None*)

Gym environment rendering

**Parameters**
**mode** (Optional[str]) – the rendering type

**Return type**
Optional[ndarray]

**reset**()

Resets the environment.

**Returns**
first observation of a new trajectory.

**Return type**
obs

**reset_infos: List[Dict[str, Any]]**

**seed**(*seed=None*)

Set the seed for the DAgger random number generator and wrapped VecEnv.

The DAgger RNG is used along with *self.beta* to determine whether the expert or robot action is forwarded to the wrapped VecEnv.

**Parameters**
**seed** (Optional[int]) – The random seed. May be None for completely random seeding.

**Return type**
List[Optional[int]]

**Returns**
A list containing the seeds for each individual env. Note that all list elements may be None, if the env does not return anything when seeded.

**set_attr**(*attr_name*, *value*, *indices=None*)

Set attribute inside vectorized environments.

**Parameters**

- **attr_name** (str) – The name of attribute to assign new value
- **value** (Any) – Value to assign to *attr_name*
- **indices** (Union[None, int, Iterable[int]]) – Indices of envs to assign value

**Return type**
None

**Returns**

**set_options**(*options=None*)

Set environment options for all environments. If a dict is passed instead of a list, the same options will be used for all environments. WARNING: Those options will only be passed to the environment at the next reset.

**Parameters**
**options** (Union[List[Dict], Dict, None]) – A dictionary of environment options to pass to each environment at the next reset.

**Return type**
None

**step**(*actions*)

Step the environments with the given action

**Parameters**
**actions** (ndarray) – the action

**Return type**
Tuple[Union[ndarray, Dict[str, ndarray], Tuple[ndarray, ...]], ndarray, ndarray, List[Dict]]

**Returns**
observation, reward, done, information

**step_async**(*actions*)

Steps with a *1 - beta* chance of using *self.get_robot_acts* instead.

DAgger needs to be able to inject imitation policy actions randomly at some subset of time steps. This method has a *self.beta* chance of keeping the *actions* passed in as an argument, and a *1 - self.beta* chance of forwarding actions generated by *self.get_robot_acts* instead. "robot" (i.e. imitation policy) action if necessary.

At the end of every episode, a *TrajectoryWithRew* is saved to *self.save_dir*, where every saved action is the expert action, regardless of whether the robot action was used during that timestep.

**Parameters**
**actions** (ndarray) – the _intended_ demonstrator/expert actions for the current state. This will be executed with probability *self.beta*. Otherwise, a "robot" (typically a BC policy) action will be sampled and executed instead via *self.get_robot_act*.

**Return type**
None

**step_wait**()

Returns observation, reward, etc after previous *step_async()* call.

Stores the transition, and saves trajectory as demo once complete.

**Return type**
Tuple[Union[ndarray, Dict[str, ndarray], Tuple[ndarray, ...]], ndarray, ndarray, List[Dict]]

**Returns**
Observation, reward, dones (is terminal?) and info dict.

**traj_accum: Optional[*TrajectoryAccumulator*]**

**property unwrapped: VecEnv**

**Return type**
VecEnv

**class** imitation.algorithms.dagger.**DAggerTrainer**(**, venv, scratch_dir, rng, beta_schedule=None, bc_trainer, custom_logger=None*)

Bases: *BaseImitationAlgorithm*

DAgger training class with low-level API suitable for interactive human feedback.

In essence, this is just BC with some helpers for incrementally resuming training and interpolating between demonstrator/learnt policies. Interaction proceeds in "rounds" in which the demonstrator first provides a fresh set of demonstrations, and then an underlying *BC* is invoked to fine-tune the policy on the entire set of demonstrations collected in all rounds so far. Demonstrations and policy/trainer checkpoints are stored in a directory with the following structure:

```
scratch-dir-name/
    checkpoint-001.pt
    checkpoint-002.pt
    …
    checkpoint-XYZ.pt
    checkpoint-latest.pt
    demos/
        round-000/
            demos_round_000_000.npz
            demos_round_000_001.npz
            …
        round-001/
            demos_round_001_000.npz
            …
        …
        round-XYZ/
            …
```

**DEFAULT_N_EPOCHS: int = 4**

The default number of BC training epochs in *extend_and_update*.

**__init__**(***, *venv*, *scratch_dir*, *rng*, *beta_schedule=None*, *bc_trainer*, *custom_logger=None*)

Builds DAggerTrainer.

**Parameters**

- **venv** (`VecEnv`) – Vectorized training environment.
- **scratch_dir** (`Union[str, bytes, PathLike]`) – Directory to use to store intermediate training information (e.g. for resuming training).
- **rng** (`Generator`) – random state for random number generation.
- **beta_schedule** (`Optional[Callable[[int], float]]`) – Provides a value of *beta* (the probability of taking expert action in any given state) at each round of training. If *None*, then *linear_beta_schedule* will be used instead.
- **bc_trainer** (*`BC`*) – A *BC* instance used to train the underlying policy.
- **custom_logger** (`Optional[`*`HierarchicalLogger`*`]`) – Where to log to; if None (default), creates a new logger.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**property batch_size: int**

**Return type**

`int`

**create_trajectory_collector**()

Create trajectory collector to extend current round's demonstration set.

**Return type**

*`InteractiveTrajectoryCollector`*

**Returns**

A collector configured with the appropriate beta, imitator policy, etc. for the current round. Refer to the documentation for *InteractiveTrajectoryCollector* to see how to use this.

**extend_and_update**(*bc_train_kwargs=None*)

Extend internal batch of data and train BC.

Specifically, this method will load new transitions (if necessary), train the model for a while, and advance the round counter. If there are no fresh demonstrations in the demonstration directory for the current round, then this will raise a *NeedsDemosException* instead of training or advancing the round counter. In that case, the user should call *.create_trajectory_collector()* and use the returned *InteractiveTrajectoryCollector* to produce a new set of demonstrations for the current interaction round.

**Parameters**

**bc_train_kwargs** (`Optional[Mapping[str, Any]]`) – Keyword arguments for calling *BC.train()*. If the *log_rollouts_venv* key is not provided, then it is set to *self.venv* by default. If neither of the *n_epochs* and *n_batches* keys are provided, then *n_epochs* is set to *self.DEFAULT_N_EPOCHS*.

**Return type**

`int`

**Returns**

New round number after advancing the round counter.

**property logger:** ***HierarchicalLogger***

Returns logger for this object.

**Return type**

*HierarchicalLogger*

**property policy: BasePolicy**

**Return type**

`BasePolicy`

**save_trainer**()

Create a snapshot of trainer in the scratch/working directory.

The created snapshot can be reloaded with *reconstruct_trainer()*. In addition to saving one copy of the policy in the trainer snapshot, this method saves a second copy of the policy in its own file. Having a second copy of the policy is convenient because it can be loaded on its own and passed to evaluation routines for other algorithms.

**Returns**

a path to one of the created *DAggerTrainer* checkpoints. policy_path: a path to one of the created *DAggerTrainer* policies.

**Return type**

checkpoint_path

**class** imitation.algorithms.dagger.**SimpleDAggerTrainer**(**, venv, scratch_dir, expert_policy, rng, expert_trajs=None, **dagger_trainer_kwargs*)

Bases: *DAggerTrainer*

Simpler subclass of DAggerTrainer for training with synthetic feedback.

**DEFAULT_N_EPOCHS: int = 4**

The default number of BC training epochs in *extend_and_update*.

**__init__**(**, venv, scratch_dir, expert_policy, rng, expert_trajs=None, **dagger_trainer_kwargs*)

Builds SimpleDAggerTrainer.

**Parameters**

- **venv** (VecEnv) – Vectorized training environment. Note that when the robot action is randomly injected (in accordance with *beta_schedule* argument), every individual environment will get a robot action simultaneously for that timestep.
- **scratch_dir** (Union[str, bytes, PathLike]) – Directory to use to store intermediate training information (e.g. for resuming training).
- **expert_policy** (BasePolicy) – The expert policy used to generate synthetic demonstrations.
- **rng** (Generator) – Random state to use for the random number generator.
- **expert_trajs** (Optional[Sequence[*Trajectory*]]) – Optional starting dataset that is inserted into the round 0 dataset.
- **dagger_trainer_kwargs** – Other keyword arguments passed to the superclass initializer *DAggerTrainer.__init__*.

**Raises**

**ValueError** – The observation or action space does not match between *venv* and *expert_policy*.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**property batch_size: int**

**Return type**

int

**create_trajectory_collector**()

Create trajectory collector to extend current round's demonstration set.

**Return type**

*InteractiveTrajectoryCollector*

**Returns**

A collector configured with the appropriate beta, imitator policy, etc. for the current round. Refer to the documentation for *InteractiveTrajectoryCollector* to see how to use this.

**extend_and_update**(*bc_train_kwargs=None*)

Extend internal batch of data and train BC.

Specifically, this method will load new transitions (if necessary), train the model for a while, and advance the round counter. If there are no fresh demonstrations in the demonstration directory for the current round, then this will raise a *NeedsDemosException* instead of training or advancing the round counter. In that case, the user should call *.create_trajectory_collector()* and use the returned *InteractiveTrajectoryCollector* to produce a new set of demonstrations for the current interaction round.

**Parameters**

**bc_train_kwargs** (Optional[Mapping[str, Any]]) – Keyword arguments for calling *BC.train()*. If the *log_rollouts_venv* key is not provided, then it is set to *self.venv* by default. If neither of the *n_epochs* and *n_batches* keys are provided, then *n_epochs* is set to *self.DEFAULT_N_EPOCHS*.

**Return type**

int

**Returns**

New round number after advancing the round counter.

**property logger:** ***HierarchicalLogger***

Returns logger for this object.

**Return type**

*HierarchicalLogger*

**property policy: BasePolicy**

**Return type**

`BasePolicy`

**save_trainer**()

Create a snapshot of trainer in the scratch/working directory.

The created snapshot can be reloaded with *reconstruct_trainer()*. In addition to saving one copy of the policy in the trainer snapshot, this method saves a second copy of the policy in its own file. Having a second copy of the policy is convenient because it can be loaded on its own and passed to evaluation routines for other algorithms.

**Returns**

a path to one of the created *DAggerTrainer* checkpoints. policy_path: a path to one of the created *DAggerTrainer* policies.

**Return type**

checkpoint_path

**train**(*total_timesteps*, ***, *rollout_round_min_episodes=3*, *rollout_round_min_timesteps=500*, *bc_train_kwargs=None*)

Train the DAgger agent.

The agent is trained in "rounds" where each round consists of a dataset aggregation step followed by BC update step.

During a dataset aggregation step, *self.expert_policy* is used to perform rollouts in the environment but there is a *1 - beta* chance (beta is determined from the round number and *self.beta_schedule*) that the DAgger agent's action is used instead. Regardless of whether the DAgger agent's action is used during the rollout, the expert action and corresponding observation are always appended to the dataset. The number of environment steps in the dataset aggregation stage is determined by the *rollout_round_min** arguments.

During a BC update step, *BC.train()* is called to update the DAgger agent on all data collected so far.

**Parameters**

- **total_timesteps** (`int`) – The number of timesteps to train inside the environment. In practice this is a lower bound, because the number of timesteps is rounded up to finish the minimum number of episodes or timesteps in the last DAgger training round, and the environment timesteps are executed in multiples of *self.venv.num_envs*.
- **rollout_round_min_episodes** (`int`) – The number of episodes the must be completed completed before a dataset aggregation step ends.
- **rollout_round_min_timesteps** (`int`) – The number of environment timesteps that must be completed before a dataset aggregation step ends. Also, that any round will always train for at least *self.batch_size* timesteps, because otherwise BC could fail to receive any batches.
- **bc_train_kwargs** (`Optional[dict]`) – Keyword arguments for calling *BC.train()*. If the *log_rollouts_venv* key is not provided, then it is set to *self.venv* by default. If neither of the *n_epochs* and *n_batches* keys are provided, then *n_epochs* is set to *self.DEFAULT_N_EPOCHS*.

**Return type**
None

## 2.15 Density-Based Reward Modeling

Density-based reward modeling is an inverse reinforcement learning (IRL) technique that assigns higher rewards to states or state-action pairs that occur more frequently in an expert's demonstrations. This variant utilizes kernel density estimation to model the underlying distribution of expert demonstrations. It assigns rewards to states or state-action pairs based on their estimated log-likelihood under the distribution of expert demonstrations.

The key intuition behind this method is to incentivize the agent to take actions that resemble the expert's actions in similar states.

While this approach is relatively simple, it does have several drawbacks:

- It assumes that the expert demonstrations are representative of the expert's behavior, which may not always be true.
- It does not provide an interpretable reward function.
- The kernel density estimation is not well-suited for high-dimensional state-action spaces.

### 2.15.1 Example

Detailed example notebook: *Learning a Reward Function using Kernel Density*

```
import pprint
import numpy as np

from stable_baselines3 import PPO
from stable_baselines3.common.policies import ActorCriticPolicy

from imitation.algorithms import density as db
from imitation.data import serialize
from imitation.util import util

rng = np.random.default_rng(0)

env = util.make_vec_env("Pendulum-v1", rng=rng, n_envs=2)
rollouts = serialize.load("../tests/testdata/expert_models/pendulum_0/rollouts/final.
↪npz")

imitation_trainer = PPO(
    ActorCriticPolicy, env, learning_rate=3e-4, gamma=0.95, ent_coef=1e-4, n_
↪steps=2048
)
density_trainer = db.DensityAlgorithm(
    venv=env,
    rng=rng,
    demonstrations=rollouts,
    rl_algo=imitation_trainer,
    density_type=db.DensityType.STATE_ACTION_DENSITY,
    is_stationary=True,
    kernel="gaussian",
    kernel_bandwidth=0.4,
    standardise_inputs=True,
)
```

(continues on next page)

(continued from previous page)

```
density_trainer.train()

def print_stats(density_trainer, n_trajectories):
    stats = density_trainer.test_policy(n_trajectories=n_trajectories)
    print("True reward function stats:")
    pprint.pprint(stats)
    stats_im = density_trainer.test_policy(true_reward=False, n_trajectories=n_
↪trajectories)
    print("Imitation reward function stats:")
    pprint.pprint(stats_im)

print("Stats before training:")
print_stats(density_trainer, 1)

density_trainer.train_policy(100)  # Train for 1_000_000 steps to approach expert␣
↪performance.

print("Stats after training:")
print_stats(density_trainer, 1)
```

### 2.15.2 API

**class** imitation.algorithms.density.**DensityAlgorithm**(***, *demonstrations*, *venv*, *rng*, *density_type=DensityType.STATE_ACTION_DENSITY*, *kernel='gaussian'*, *kernel_bandwidth=0.5*, *rl_algo=None*, *is_stationary=True*, *standardise_inputs=True*, *custom_logger=None*, *allow_variable_horizon=False*)

Bases: *DemonstrationAlgorithm*

Learns a reward function based on density modeling.

Specifically, it constructs a non-parametric estimate of *p(s)*, *p(s,a)*, *p(s,s')* and then computes a reward using the log of these probabilities.

**__init__**(***, *demonstrations*, *venv*, *rng*, *density_type=DensityType.STATE_ACTION_DENSITY*, *kernel='gaussian'*, *kernel_bandwidth=0.5*, *rl_algo=None*, *is_stationary=True*, *standardise_inputs=True*, *custom_logger=None*, *allow_variable_horizon=False*)

Builds DensityAlgorithm.

**Parameters**

- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*, None]) – expert demonstration trajectories.
- **density_type** (*DensityType*) – type of density to train on: single state, state-action pairs, or state-state pairs.
- **kernel** (str) – kernel to use for density estimation with *sklearn.KernelDensity*.
- **kernel_bandwidth** (float) – bandwidth of kernel. If *standardise_inputs* is true and you are using a Gaussian kernel, then it probably makes sense to set this somewhere between 0.1 and 1.

- **venv** (VecEnv) – The environment to learn a reward model in. We don't actually need any environment interaction to fit the reward model, but we use this to extract the observation and action space, and to train the RL algorithm *rl_algo* (if specified).
- **rng** (Generator) – random state for sampling from demonstrations.
- **rl_algo** (Optional[BaseAlgorithm]) – An RL algorithm to train on the resulting reward model (optional).
- **is_stationary** (bool) – if True, share same density models for all timesteps; if False, use a different density model for each timestep. A non-stationary model is particularly likely to be useful when using STATE_DENSITY, to encourage agent to imitate entire trajectories, not just a few states that have high frequency in the demonstration dataset. If non-stationary, demonstrations must be trajectories, not transitions (which do not contain timesteps).
- **standardise_inputs** (bool) – if True, then the inputs to the reward model will be standardised to have zero mean and unit variance over the demonstration trajectories. Otherwise, inputs will be passed to the reward model with their ordinary scale.
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **allow_variable_horizon** (bool) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/guide/variable_horizon.html before overriding this.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**buffering_wrapper:** ***BufferingWrapper***

**density_type:** ***DensityType***

**is_stationary: bool**

**kernel: str**

**kernel_bandwidth: float**

**property logger:** ***HierarchicalLogger***

**Return type**

*HierarchicalLogger*

**property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**

BasePolicy

**rl_algo: Optional[BaseAlgorithm]**

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

**Return type**

None

**standardise: bool**

**test_policy**(**, n_trajectories=10, true_reward=True*)

Test current imitation policy on environment & give some rollout stats.

**Parameters**

- **n_trajectories** (int) – number of rolled-out trajectories.
- **true_reward** (bool) – should this use ground truth reward from underlying environment (True), or imitation reward (False)?

**Returns**

**rollout statistics collected by** *imitation.utils.rollout.rollout_stats()*.

**Return type**

dict

**train**()

Fits the density model to demonstration data *self.transitions*.

**Return type**

None

**train_policy**(*n_timesteps=1000000, **kwargs*)

Train the imitation policy for a given number of timesteps.

**Parameters**

- **n_timesteps** (int) – number of timesteps to train the policy for.
- **kwargs** (*dict*) – extra arguments that will be passed to the *learn()* method of the imitation RL model. Refer to Stable Baselines docs for details.

**Return type**

None

**transitions: Dict[Optional[int], ndarray]**

**venv: VecEnv**

**venv_wrapped:** ***RewardVecEnvWrapper***

**wrapper_callback:** ***WrappedRewardCallback***

## 2.16 Maximum Causal Entropy Inverse Reinforcement Learning (MCE IRL)

Implements Modeling Interaction via the Principle of Maximum Causal Entropy.

### 2.16.1 Example

Detailed example notebook: *Learn a Reward Function using Maximum Conditional Entropy Inverse Reinforcement Learning*

```
from functools import partial

from seals import base_envs
from seals.diagnostics.cliff_world import CliffWorldEnv
import numpy as np

from stable_baselines3.common.vec_env import DummyVecEnv

from imitation.algorithms.mce_irl import (
    MCEIRL,
    mce_occupancy_measures,
    mce_partition_fh,
)
from imitation.data import rollout
from imitation.rewards import reward_nets

rng = np.random.default_rng(0)

env_creator = partial(CliffWorldEnv, height=4, horizon=8, width=7, use_xy_obs=True)
env_single = env_creator()

state_env_creator = lambda: base_envs.ExposePOMDPStateWrapper(env_creator())

# This is just a vectorized environment because `generate_trajectories` expects one
state_venv = DummyVecEnv([state_env_creator] * 4)

_, _, pi = mce_partition_fh(env_single)

_, om = mce_occupancy_measures(env_single, pi=pi)

reward_net = reward_nets.BasicRewardNet(
    env_single.observation_space,
    env_single.action_space,
    hid_sizes=[256],
    use_action=False,
    use_done=False,
    use_next_state=False,
)

# training on analytically computed occupancy measures
mce_irl = MCEIRL(
    om,
    env_single,
    reward_net,
    log_interval=250,
    optimizer_kwargs={"lr": 0.01},
    rng=rng,
)
occ_measure = mce_irl.train()

imitation_trajs = rollout.generate_trajectories(
    policy=mce_irl.policy,
    venv=state_venv,
```

(continues on next page)

(continued from previous page)

```
    sample_until=rollout.make_min_timesteps(5000),
    rng=rng,
)
print("Imitation stats: ", rollout.rollout_stats(imitation_trajs))
```

### 2.16.2 API

**class** imitation.algorithms.mce_irl.**MCEIRL**(*demonstrations*, *env*, *reward_net*, *rng*, *optimizer_cls=<class 'torch.optim.adam.Adam'>*, *optimizer_kwargs=None*, *discount=1.0*, *linf_eps=0.001*, *grad_l2_eps=0.0001*, *log_interval=100*, ***, *custom_logger=None*)

Bases: *DemonstrationAlgorithm*[*TransitionsMinimal*]

Tabular MCE IRL.

Reward is a function of observations, but policy is a function of states.

The "observations" effectively exist just to let MCE IRL learn a reward in a reasonable feature space, giving a helpful inductive bias, e.g. that similar states have similar reward.

Since we are performing planning to compute the policy, there is no need for function approximation in the policy.

**__init__**(*demonstrations*, *env*, *reward_net*, *rng*, *optimizer_cls=<class 'torch.optim.adam.Adam'>*, *optimizer_kwargs=None*, *discount=1.0*, *linf_eps=0.001*, *grad_l2_eps=0.0001*, *log_interval=100*, ***, *custom_logger=None*)

Creates MCE IRL.

**Parameters**

- **demonstrations** (Union[ndarray, Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*, None]) – Demonstrations from an expert (optional). Can be a sequence of trajectories, or transitions, an iterable over mappings that represent a batch of transitions, or a state occupancy measure. The demonstrations must have observations one-hot coded unless demonstrations is a state-occupancy measure.
- **env** (TabularModelPOMDP) – a tabular MDP.
- **rng** (Generator) – random state used for sampling from policy.
- **reward_net** (*RewardNet*) – a neural network that computes rewards for the supplied observations.
- **optimizer_cls** (Type[Optimizer]) – optimizer to use for supervised training.
- **optimizer_kwargs** (Optional[Mapping[str, Any]]) – keyword arguments for optimizer construction.
- **discount** (float) – the discount factor to use when computing occupancy measure. If not 1.0 (undiscounted), then *demonstrations* must either be a (discounted) state-occupancy measure, or trajectories. Transitions are *not allowed* as we cannot discount them appropriately without knowing the timestep they were drawn from.
- **linf_eps** (float) – optimisation terminates if the $l_{infty}$ distance between the demonstrator's state occupancy measure and the state occupancy measure for the current reward falls below this value.

- **grad_l2_eps** (float) – optimisation also terminates if the $ell_2$ norm of the MCE IRL gradient falls below this value.
- **log_interval** (Optional[int]) – how often to log current loss stats (using *logging*). None to disable.
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.

**Raises**

**ValueError** – if the env horizon is not finite (or an integer).

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**demo_state_om: Optional[ndarray]**

**property logger:** ***HierarchicalLogger***

**Return type**

*HierarchicalLogger*

**property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**

BasePolicy

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (Union[ndarray, Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**

None

**train**(*max_iter=1000*)

Runs MCE IRL.

**Parameters**

**max_iter** (int) – The maximum number of iterations to train for. May terminate earlier if *self.linf_eps* or *self.grad_l2_eps* thresholds are reached.

**Return type**

ndarray

**Returns**

State occupancy measure for the final reward function. *self.reward_net* and *self.optimizer* will be updated in-place during optimisation.

**class** imitation.algorithms.base.**DemonstrationAlgorithm**(**, demonstrations, custom_logger=None, allow_variable_horizon=False*)

Bases: *BaseImitationAlgorithm*, Generic[TransitionKind]

An algorithm that learns from demonstration: BC, IRL, etc.

**`__init__`**(***, *demonstrations*, *custom_logger=None*, *allow_variable_horizon=False*)

Creates an algorithm that learns from demonstrations.

**Parameters**

- **demonstrations** (`Union[Iterable[`*`Trajectory`*`], Iterable[`*`TransitionMapping`*`], `*`TransitionsMinimal`*`, None]`) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **custom_logger** (`Optional[`*`HierarchicalLogger`*`]`) – Where to log to; if None (default), creates a new logger.
- **allow_variable_horizon** (`bool`) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/getting-started/variable-horizon.html before overriding this.

**`allow_variable_horizon: bool`**

If True, allow variable horizon trajectories; otherwise error if detected.

**`abstract property policy: BasePolicy`**

Returns a policy imitating the demonstration data.

**Return type**

`BasePolicy`

**`abstract set_demonstrations`**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (`Union[Iterable[`*`Trajectory`*`], Iterable[`*`TransitionMapping`*`], `*`TransitionsMinimal`*`]`) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**

`None`

## 2.17 Preference Comparisons

The preference comparison algorithm learns a reward function from preferences between pairs of trajectories. The comparisons are modeled as being generated from a Bradley-Terry (or Boltzmann rational) model, where the probability of preferring trajectory A over B is proportional to the exponential of the difference between the return of trajectory A minus B. In other words, the difference in returns forms a logit for a binary classification problem, and accordingly the reward function is trained using a cross-entropy loss to predict the preference comparison.

**Note:**

- Our implementation is based on the Deep Reinforcement Learning from Human Preferences algorithm.
- An ensemble of reward networks can also be trained instead of a single network. The uncertainty in the preference between the member networks can be used to actively select preference queries.

### 2.17.1 Example

You can copy this example to train PPO on Pendulum using a reward model trained on 200 synthetic preference comparisons. For a more detailed example, refer to *Learning a Reward Function using Preference Comparisons*.

```python
import numpy as np

from stable_baselines3 import PPO
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3.ppo import MlpPolicy

from imitation.algorithms import preference_comparisons
from imitation.policies.base import FeedForward32Policy, NormalizeFeaturesExtractor
from imitation.rewards.reward_nets import BasicRewardNet
from imitation.rewards.reward_wrapper import RewardVecEnvWrapper
from imitation.util.networks import RunningNorm
from imitation.util.util import make_vec_env

rng = np.random.default_rng(0)

venv = make_vec_env("Pendulum-v1", rng=rng)

reward_net = BasicRewardNet(
    venv.observation_space, venv.action_space, normalize_input_layer=RunningNorm,
)

fragmenter = preference_comparisons.RandomFragmenter(warning_threshold=0, rng=rng)
gatherer = preference_comparisons.SyntheticGatherer(rng=rng)
preference_model = preference_comparisons.PreferenceModel(reward_net)
reward_trainer = preference_comparisons.BasicRewardTrainer(
    preference_model=preference_model,
    loss=preference_comparisons.CrossEntropyRewardLoss(),
    epochs=10,
    rng=rng,
)

agent = PPO(
    policy=FeedForward32Policy,
    policy_kwargs=dict(
        features_extractor_class=NormalizeFeaturesExtractor,
        features_extractor_kwargs=dict(normalize_class=RunningNorm),
    ),
    env=venv,
    n_steps=2048 // venv.num_envs,
    clip_range=0.1,
    ent_coef=0.01,
    gae_lambda=0.95,
    n_epochs=10,
    gamma=0.97,
    learning_rate=2e-3,
)

trajectory_generator = preference_comparisons.AgentTrainer(
    algorithm=agent,
    reward_fn=reward_net,
```

(continues on next page)

(continued from previous page)

```
    venv=venv,
    exploration_frac=0.05,
    rng=rng,
)

pref_comparisons = preference_comparisons.PreferenceComparisons(
    trajectory_generator,
    reward_net,
    num_iterations=5, # Set to 60 for better performance
    fragmenter=fragmenter,
    preference_gatherer=gatherer,
    reward_trainer=reward_trainer,
    initial_epoch_multiplier=4,
    initial_comparison_frac=0.1,
    query_schedule="hyperbolic",
)
pref_comparisons.train(total_timesteps=50_000, total_comparisons=200)

n_eval_episodes = 10
reward_mean, reward_std = evaluate_policy(agent.policy, venv, n_eval_episodes)
reward_stderr = reward_std/np.sqrt(n_eval_episodes)
print(f"Reward: {reward_mean:.0f} +/- {reward_stderr:.0f}")
```

### 2.17.2 API

**class** imitation.algorithms.preference_comparisons.**PreferenceComparisons**(*trajectory_generator*, *reward_model*, *num_iterations*, *fragmenter=None*, *preference_gatherer=None*, *reward_trainer=None*, *comparison_queue_size=None*, *fragment_length=100*, *transition_oversampling=1*, *initial_comparison_frac=0.1*, *initial_epoch_multiplier=200.0*, *custom_logger=None*, *allow_variable_horizon=False*, *rng=None*, *query_schedule='hyperbolic'*)

Bases: `BaseImitationAlgorithm`

Main interface for reward learning using preference comparisons.

**__init__**(*trajectory_generator*, *reward_model*, *num_iterations*, *fragmenter=None*, *preference_gatherer=None*, *reward_trainer=None*, *comparison_queue_size=None*, *fragment_length=100*, *transition_oversampling=1*, *initial_comparison_frac=0.1*, *initial_epoch_multiplier=200.0*, *custom_logger=None*, *allow_variable_horizon=False*, *rng=None*, *query_schedule='hyperbolic'*)

Initialize the preference comparison trainer.

The loggers of all subcomponents are overridden with the logger used by this class.

**Parameters**

- **trajectory_generator** (*TrajectoryGenerator*) – generates trajectories while optionally training an RL agent on the learned reward function (can also be a sampler from a static dataset of trajectories though).
- **reward_model** (*RewardNet*) – a RewardNet instance to be used for learning the reward
- **num_iterations** (int) – number of times to train the agent against the reward model and then train the reward model against newly gathered preferences.
- **fragmenter** (Optional[*Fragmenter*]) – takes in a set of trajectories and returns pairs of fragments for which preferences will be gathered. These fragments could be random, or they could be selected more deliberately (active learning). Default is a random fragmenter.
- **preference_gatherer** (Optional[*PreferenceGatherer*]) – how to get preferences between trajectory fragments. Default (and currently the only option) is to use synthetic preferences based on ground-truth rewards. Human preferences could be implemented here in the future.
- **reward_trainer** (Optional[*RewardTrainer*]) – trains the reward model based on pairs of fragments and associated preferences. Default is to use the preference model and loss function from DRLHP.
- **comparison_queue_size** (Optional[int]) – the maximum number of comparisons to keep in the queue for training the reward model. If None, the queue will grow without bound as new comparisons are added.
- **fragment_length** (int) – number of timesteps per fragment that is used to elicit preferences
- **transition_oversampling** (float) – factor by which to oversample transitions before creating fragments. Since fragments are sampled with replacement, this is usually chosen > 1 to avoid having the same transition in too many fragments.
- **initial_comparison_frac** (float) – fraction of the total_comparisons argument to train() that will be sampled before the rest of training begins (using a randomly initialized agent). This can be used to pretrain the reward model before the agent is trained on the learned reward, to help avoid irreversibly learning a bad policy from an untrained reward. Note that there will often be some additional pretraining comparisons since *comparisons_per_iteration* won't exactly divide the total number of comparisons. How many such comparisons there are depends discontinuously on *total_comparisons* and *comparisons_per_iteration*.
- **initial_epoch_multiplier** (float) – before agent training begins, train the reward model for this many more epochs than usual (on fragments sampled from a random agent).
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **allow_variable_horizon** (bool) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/guide/variable_horizon.html before overriding this.
- **rng** (Optional[Generator]) – random number generator to use for initializing subcomponents such as fragmenter. Only used when default components are used; if you instantiate your own fragmenter, preference gatherer, etc., you are responsible for seeding them!
- **query_schedule** (Union[str, Callable[[float], float]]) – one of ("constant", "hyperbolic", "inverse_quadratic"), or a function that takes in a float between 0 and 1 inclu-

sive, representing a fraction of the total number of timesteps elapsed up to some time T, and returns a potentially unnormalized probability indicating the fraction of *total_comparisons* that should be queried at that iteration. This function will be called *num_iterations* times in *__init__()* with values from *np.linspace(0, 1, num_iterations)* as input. The outputs will be normalized to sum to 1 and then used to apportion the comparisons among the *num_iterations* iterations.

**Raises**

`ValueError` – if *query_schedule* is not a valid string or callable.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**property logger:** ***HierarchicalLogger***

**Return type**

*HierarchicalLogger*

**train**(*total_timesteps*, *total_comparisons*, *callback=None*)

Train the reward model and the policy if applicable.

**Parameters**

- **total_timesteps** (`int`) – number of environment interaction steps
- **total_comparisons** (`int`) – number of preferences to gather in total
- **callback** (`Optional[Callable[[int], None]]`) – callback functions called at the end of each iteration

**Return type**

`Mapping[str, Any]`

**Returns**

A dictionary with final metrics such as loss and accuracy of the reward model.

**class** imitation.algorithms.base.**BaseImitationAlgorithm**(**, *custom_logger=None*, *allow_variable_horizon=False*)

Bases: `ABC`

Base class for all imitation learning algorithms.

**__init__**(**, *custom_logger=None*, *allow_variable_horizon=False*)

Creates an imitation learning algorithm.

**Parameters**

- **custom_logger** (`Optional[`*HierarchicalLogger*`]`) – Where to log to; if None (default), creates a new logger.
- **allow_variable_horizon** (`bool`) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/getting-started/variable-horizon.html before overriding this.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**property logger:** *HierarchicalLogger*

**Return type**
*HierarchicalLogger*

## 2.18 Soft Q Imitation Learning (SQIL)

Soft Q Imitation learning learns to imitate a policy from demonstrations by using the DQN algorithm with modified rewards. During each policy update, half of the batch is sampled from the demonstrations and half is sampled from the environment. Expert demonstrations are assigned a reward of 1, and the environment is assigned a reward of 0. This encourages the policy to imitate the demonstrations, and to simultaneously avoid states not seen in the demonstrations.

**Note:** This implementation is based on the DQN implementation in Stable Baselines 3, which does not implement the soft Q-learning and therefore does not support continuous actions. Therefore, this implementation only supports discrete actions and the name "soft" Q-learning could be misleading.

### 2.18.1 Example

Detailed example notebook: *Train an Agent using Soft Q Imitation Learning*

```
import datasets
import gymnasium as gym
from stable_baselines3.common.evaluation import evaluate_policy
from stable_baselines3.common.vec_env import DummyVecEnv

from imitation.algorithms import sqil
from imitation.data import huggingface_utils

# Download some expert trajectories from the HuggingFace Datasets Hub.
dataset = datasets.load_dataset("HumanCompatibleAI/ppo-CartPole-v1")
rollouts = huggingface_utils.TrajectoryDatasetSequence(dataset["train"])

sqil_trainer = sqil.SQIL(
    venv=DummyVecEnv([lambda: gym.make("CartPole-v1")]),
    demonstrations=rollouts,
    policy="MlpPolicy",
)
# Hint: set to 1_000_000 to match the expert performance.
sqil_trainer.train(total_timesteps=1_000)
reward, _ = evaluate_policy(sqil_trainer.policy, sqil_trainer.venv, 10)
print("Reward:", reward)
```

### 2.18.2 API

**class** imitation.algorithms.sqil.**SQIL**(**, venv, demonstrations, policy, custom_logger=None, rl_algo_class=<class 'stable_baselines3.dqn.dqn.DQN'>, rl_kwargs=None*)

Bases: *DemonstrationAlgorithm*[*Transitions*]

Soft Q Imitation Learning (SQIL).

Trains a policy via DQN-style Q-learning, replacing half the buffer with expert demonstrations and adjusting the rewards.

**__init__**(**, venv, demonstrations, policy, custom_logger=None, rl_algo_class=<class 'stable_baselines3.dqn.dqn.DQN'>, rl_kwargs=None*)

Builds SQIL.

**Parameters**

- **venv** (VecEnv) – The vectorized environment to train on.
- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*, None]) – Demonstrations to use for training.
- **policy** (Union[str, Type[BasePolicy]]) – The policy model to use (SB3).
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **rl_algo_class** (Type[OffPolicyAlgorithm]) – Off-policy RL algorithm to use.
- **rl_kwargs** (Optional[Dict[str, Any]]) – Keyword arguments to pass to the RL algorithm constructor.

**Raises**

**ValueError** – if *dqn_kwargs* includes a key *replay_buffer_class* or *replay_buffer_kwargs*.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**expert_buffer: ReplayBuffer**

**property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**

BasePolicy

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**

None

**train** (*, *total_timesteps*, *tb_log_name='SQIL'*, ***kwargs*)

download this notebook here

## 2.19 Train an Agent using Behavior Cloning

Behavior cloning is the most naive approach to imitation learning. We take the transitions of trajectories taken by some expert and use them as training samples to train a new policy. The method has many drawbacks and often does not work. However in this example, where we use an agent for the seals/CartPole-v0 environment, it is feasible.

Note that we use a variant of the CartPole environment from the seals package, which has fixed episode durations. Read more about why we do this here.

First we need some kind of expert in CartPole so we can sample some expert trajectories. For convenience we just download one from the HuggingFace model hub.

If you want to train an expert yourself have a look at the training documenation of RL Baselines3 Zoo.

```python
import numpy as np
import gymnasium as gym
from imitation.policies.serialize import load_policy
from imitation.util.util import make_vec_env
from imitation.data.wrappers import RolloutInfoWrapper

env = make_vec_env(
    "seals:seals/CartPole-v0",
    rng=np.random.default_rng(),
    post_wrappers=[
        lambda env, _: RolloutInfoWrapper(env)
    ],  # needed for computing rollouts later
)
expert = load_policy(
    "ppo-huggingface",
    organization="HumanCompatibleAI",
    env_name="seals/CartPole-v0",
    venv=env,
)
```

Let's quickly check if the expert is any good. We usually should be able to reach a reward of 500, which is the maximum achievable value.

```python
from stable_baselines3.common.evaluation import evaluate_policy

reward, _ = evaluate_policy(expert, env, 10)
print(reward)
```

```
500.0
```

Now we can use the expert to sample some trajectories. We flatten them right away since we are only interested in the individual transitions for behavior cloning. `imitation` comes with a number of helper functions that makes collecting those transitions really easy. First we collect 50 episode rollouts, then we flatten them to just the transitions that we need for training.

Note that the rollout function requires a vectorized environment and needs the `RolloutInfoWrapper` around each of the environments. This is why we passed the `post_wrappers` argument to `make_vec_env` above.

```
from imitation.data import rollout

rng = np.random.default_rng()
rollouts = rollout.rollout(
    expert,
    env,
    rollout.make_sample_until(min_timesteps=None, min_episodes=50),
    rng=rng,
)
transitions = rollout.flatten_trajectories(rollouts)
```

Let's have a quick look at what we just generated using those library functions:

```
print(
    f"""The `rollout` function generated a list of {len(rollouts)} {type(rollouts[0])}
↪.
After flattening, this list is turned into a {type(transitions)} object containing
↪{len(transitions)} transitions.
The transitions object contains arrays for: {', '.join(transitions.__dict__.keys())}."
"""
)
```

```
The `rollout` function generated a list of 56 <class 'imitation.data.types.
↪TrajectoryWithRew'>.
After flattening, this list is turned into a <class 'imitation.data.types.Transitions
↪'> object containing 28000 transitions.
The transitions object contains arrays for: obs, acts, infos, next_obs, dones."
```

After we collected our transitions, it's time to set up our behavior cloning algorithm.

```
from imitation.algorithms import bc

bc_trainer = bc.BC(
    observation_space=env.observation_space,
    action_space=env.action_space,
    demonstrations=transitions,
    rng=rng,
)
```

As you can see the untrained policy only gets poor rewards:

```
reward_before_training, _ = evaluate_policy(bc_trainer.policy, env, 10)
print(f"Reward before training: {reward_before_training}")
```

```
Reward before training: 8.2
```

After training, we can match the rewards of the expert (500):

```
bc_trainer.train(n_epochs=1)
reward_after_training, _ = evaluate_policy(bc_trainer.policy, env, 10)
print(f"Reward after training: {reward_after_training}")
```

```
---------------------------------
| batch_size          | 32        |
| bc/                 |           |
|    batch            | 0         |
```

(continues on next page)

(continued from previous page)

```
|    ent_loss       | -0.000693 |
|    entropy        | 0.693     |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 72.5      |
|    loss           | 0.693     |
|    neglogp        | 0.694     |
|    prob_true_act  | 0.5       |
|    samples_so_far | 32        |
---------------------------------
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 500       |
|    ent_loss       | -0.000237 |
|    entropy        | 0.237     |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 93.5      |
|    loss           | 0.321     |
|    neglogp        | 0.321     |
|    prob_true_act  | 0.832     |
|    samples_so_far | 16032     |
---------------------------------
Reward after training: 500.0
```

download this notebook here

## 2.20 Train an Agent using the DAgger Algorithm

The DAgger algorithm is an extension of behavior cloning. In behavior cloning, the training trajectories are recorded directly from an expert. In DAgger, the learner generates the trajectories but an expert corrects the actions with the optimal actions in each of the visited states. This ensures that the state distribution of the training data matches that of the learner's current policy.

First we need an expert to learn from. For convenience we download one from the HuggingFace model hub.

```
import numpy as np
import gymnasium as gym
from imitation.policies.serialize import load_policy
from imitation.util.util import make_vec_env

env = make_vec_env(
    "seals:seals/CartPole-v0",
    rng=np.random.default_rng(),
    n_envs=1,
)
expert = load_policy(
    "ppo-huggingface",
    organization="HumanCompatibleAI",
    env_name="seals/CartPole-v0",
    venv=env,
)
```

Then we can construct a DAgger trainer und use it to train the policy on the cartpole environment.

```
import tempfile

from imitation.algorithms import bc
from imitation.algorithms.dagger import SimpleDAggerTrainer

bc_trainer = bc.BC(
    observation_space=env.observation_space,
    action_space=env.action_space,
    rng=np.random.default_rng(),
)

with tempfile.TemporaryDirectory(prefix="dagger_example_") as tmpdir:
    print(tmpdir)
    dagger_trainer = SimpleDAggerTrainer(
        venv=env,
        scratch_dir=tmpdir,
        expert_policy=expert,
        bc_trainer=bc_trainer,
        rng=np.random.default_rng(),
    )

    dagger_trainer.train(2000)
```

```
/tmp/dagger_example_aroskjbg
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -0.000693 |
|    entropy        | 0.693     |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 72.5      |
|    loss           | 0.692     |
|    neglogp        | 0.692     |
|    prob_true_act  | 0.5       |
|    samples_so_far | 32        |
| rollout/          |           |
|    return_max     | 23        |
|    return_mean    | 16.8      |
|    return_min     | 9         |
|    return_std     | 4.75      |
---------------------------------
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -0.000362 |
|    entropy        | 0.362     |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 86.6      |
|    loss           | 0.305     |
|    neglogp        | 0.305     |
|    prob_true_act  | 0.78      |
|    samples_so_far | 32        |
| rollout/          |           |
```

(continues on next page)

(continued from previous page)

```
|    return_max      | 154       |
|    return_mean     | 83.2      |
|    return_min      | 51        |
|    return_std      | 37.4      |
---------------------------------
```

Finally, the evaluation shows, that we actually trained a policy that solves the environment (500 is the max reward).

```
from stable_baselines3.common.evaluation import evaluate_policy

reward, _ = evaluate_policy(dagger_trainer.policy, env, 20)
print(reward)
```

```
500.0
```

download this notebook here

## 2.21 Train an Agent using Generative Adversarial Imitation Learning

The idea of generative adversarial imitation learning is to train a discriminator network to distinguish between expert trajectories and learner trajectories. The learner is trained using a traditional reinforcement learning algorithm such as PPO and is rewarded for trajectories that make the discriminator think that it was an expert trajectory.

As usual, we first need an expert. Again, we download one from the HuggingFace model hub for convenience.

Note that we use a variant of the CartPole environment from the seals package, which has fixed episode durations. Read more about why we do this here.

```
import numpy as np
from imitation.policies.serialize import load_policy
from imitation.util.util import make_vec_env
from imitation.data.wrappers import RolloutInfoWrapper

SEED = 42

env = make_vec_env(
    "seals:seals/CartPole-v0",
    rng=np.random.default_rng(SEED),
    n_envs=8,
    post_wrappers=[
        lambda env, _: RolloutInfoWrapper(env)
    ],  # needed for computing rollouts later
)
expert = load_policy(
    "ppo-huggingface",
    organization="HumanCompatibleAI",
    env_name="seals/CartPole-v0",
    venv=env,
)
```

We generate some expert trajectories, that the discriminator needs to distinguish from the learner's trajectories.

```
from imitation.data import rollout

```

(continues on next page)

(continued from previous page)

```
rollouts = rollout.rollout(
    expert,
    env,
    rollout.make_sample_until(min_timesteps=None, min_episodes=60),
    rng=np.random.default_rng(SEED),
)
```

Now we are ready to set up our GAIL trainer. Note, that the `reward_net` is actually the network of the discriminator. We evaluate the learner before and after training so we can see if it made any progress.

First we construct a GAIL trainer …

```
from imitation.algorithms.adversarial.gail import GAIL
from imitation.rewards.reward_nets import BasicRewardNet
from imitation.util.networks import RunningNorm
from stable_baselines3 import PPO
from stable_baselines3.ppo import MlpPolicy
from stable_baselines3.common.evaluation import evaluate_policy

learner = PPO(
    env=env,
    policy=MlpPolicy,
    batch_size=64,
    ent_coef=0.0,
    learning_rate=0.0004,
    gamma=0.95,
    n_epochs=5,
    seed=SEED,
)
reward_net = BasicRewardNet(
    observation_space=env.observation_space,
    action_space=env.action_space,
    normalize_input_layer=RunningNorm,
)
gail_trainer = GAIL(
    demonstrations=rollouts,
    demo_batch_size=1024,
    gen_replay_buffer_capacity=512,
    n_disc_updates_per_round=8,
    venv=env,
    gen_algo=learner,
    reward_net=reward_net,
)
```

… then we evaluate it before training …

```
env.seed(SEED)
learner_rewards_before_training, _ = evaluate_policy(
    learner, env, 100, return_episode_rewards=True
)
```

… and train it …

```
gail_trainer.train(200_000)
```

```
------------------------------------------
| raw/                        |          |
|    gen/rollout/ep_len_mean  | 500      |
|    gen/rollout/ep_rew_mean  | 29.8     |
|    gen/time/fps             | 6266     |
|    gen/time/iterations      | 1        |
|    gen/time/time_elapsed    | 2        |
|    gen/time/total_timesteps | 16384    |
------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.696    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.694    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.693    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.69     |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
```

(continues on next page)

(continued from previous page)

```
|    disc/global_step                | 1        |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.5      |
|    disc/disc_acc_expert            | 0        |
|    disc/disc_acc_gen               | 1        |
|    disc/disc_entropy               | 0.69     |
|    disc/disc_loss                  | 0.688    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 1        |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.5      |
|    disc/disc_acc_expert            | 0        |
|    disc/disc_acc_gen               | 1        |
|    disc/disc_entropy               | 0.69     |
|    disc/disc_loss                  | 0.686    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 1        |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.5      |
|    disc/disc_acc_expert            | 0        |
|    disc/disc_acc_gen               | 1        |
|    disc/disc_entropy               | 0.69     |
|    disc/disc_loss                  | 0.684    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 1        |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.5      |
|    disc/disc_acc_expert            | 0        |
|    disc/disc_acc_gen               | 1        |
|    disc/disc_entropy               | 0.69     |
|    disc/disc_loss                  | 0.683    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 1        |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
| mean/                               |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.689    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
|    gen/rollout/ep_len_mean          | 500      |
|    gen/rollout/ep_rew_mean          | 29.8     |
|    gen/time/fps                     | 6.27e+03 |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 2        |
|    gen/time/total_timesteps         | 1.64e+04 |
|    gen/train/approx_kl              | 0.00905  |
|    gen/train/clip_fraction          | 0.0295   |
|    gen/train/clip_range             | 0.2      |
|    gen/train/entropy_loss           | -0.686   |
|    gen/train/explained_variance     | 0.0301   |
|    gen/train/learning_rate          | 0.0004   |
|    gen/train/loss                   | 0.127    |
|    gen/train/n_updates              | 5        |
|    gen/train/policy_gradient_loss   | -0.0015  |
|    gen/train/value_loss             | 4.43     |
--------------------------------------------------
----------------------------------------------------
| raw/                               |             |
|    gen/rollout/ep_len_mean         | 500         |
|    gen/rollout/ep_rew_mean         | 31.9        |
|    gen/rollout/ep_rew_wrapped_mean | 268         |
|    gen/time/fps                    | 6260        |
|    gen/time/iterations             | 1           |
|    gen/time/time_elapsed           | 2           |
|    gen/time/total_timesteps        | 32768       |
|    gen/train/approx_kl             | 0.009048736 |
|    gen/train/clip_fraction         | 0.0295      |
|    gen/train/clip_range            | 0.2         |
|    gen/train/entropy_loss          | -0.686      |
|    gen/train/explained_variance    | 0.0301      |
|    gen/train/learning_rate         | 0.0004      |
|    gen/train/loss                  | 0.127       |
|    gen/train/n_updates             | 5           |
|    gen/train/policy_gradient_loss  | -0.0015     |
|    gen/train/value_loss            | 4.43        |
----------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.691    |
|    disc/disc_loss                   | 0.685    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
```

(continues on next page)

(continued from previous page)

```
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.691    |
|    disc/disc_loss                   | 0.684    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.691    |
|    disc/disc_loss                   | 0.683    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.691    |
|    disc/disc_loss                   | 0.682    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.691    |
|    disc/disc_loss                   | 0.68     |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.691    |
|    disc/disc_loss                   | 0.68     |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.691    |
|    disc/disc_loss                   | 0.679    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.691    |
|    disc/disc_loss                   | 0.678    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                               |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.691    |
|    disc/disc_loss                   | 0.681    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
|    gen/rollout/ep_len_mean          | 500      |
|    gen/rollout/ep_rew_mean          | 31.9     |
|    gen/rollout/ep_rew_wrapped_mean  | 268      |
|    gen/time/fps                     | 6.26e+03 |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 2        |
|    gen/time/total_timesteps         | 3.28e+04 |
```

(continues on next page)

(continued from previous page)

```
|    gen/train/approx_kl              | 0.0102   |
|    gen/train/clip_fraction          | 0.133    |
|    gen/train/clip_range             | 0.2      |
|    gen/train/entropy_loss           | -0.686   |
|    gen/train/explained_variance     | 0.841    |
|    gen/train/learning_rate          | 0.0004   |
|    gen/train/loss                   | 0.0145   |
|    gen/train/n_updates              | 10       |
|    gen/train/policy_gradient_loss   | -0.00786 |
|    gen/train/value_loss             | 0.248    |
--------------------------------------------------
----------------------------------------------------
| raw/                                |             |
|    gen/rollout/ep_len_mean         | 500         |
|    gen/rollout/ep_rew_mean         | 34.1        |
|    gen/rollout/ep_rew_wrapped_mean | 275         |
|    gen/time/fps                    | 5998        |
|    gen/time/iterations             | 1           |
|    gen/time/time_elapsed           | 2           |
|    gen/time/total_timesteps        | 49152       |
|    gen/train/approx_kl             | 0.010180451 |
|    gen/train/clip_fraction         | 0.133       |
|    gen/train/clip_range            | 0.2         |
|    gen/train/entropy_loss          | -0.686      |
|    gen/train/explained_variance    | 0.841       |
|    gen/train/learning_rate         | 0.0004      |
|    gen/train/loss                  | 0.0145      |
|    gen/train/n_updates             | 10          |
|    gen/train/policy_gradient_loss  | -0.00786    |
|    gen/train/value_loss            | 0.248       |
----------------------------------------------------
--------------------------------------------------
| raw/                                |           |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.672    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |           |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.671    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.67     |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.668    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.667    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.668    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_entropy                | 0.689    |
|    disc/disc_loss                   | 0.664    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.689    |
|    disc/disc_loss                   | 0.661    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                               |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.69     |
|    disc/disc_loss                   | 0.668    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
|    gen/rollout/ep_len_mean          | 500      |
|    gen/rollout/ep_rew_mean          | 34.1     |
|    gen/rollout/ep_rew_wrapped_mean  | 275      |
|    gen/time/fps                     | 6e+03    |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 2        |
|    gen/time/total_timesteps         | 4.92e+04 |
|    gen/train/approx_kl              | 0.0153   |
|    gen/train/clip_fraction          | 0.195    |
|    gen/train/clip_range             | 0.2      |
|    gen/train/entropy_loss           | -0.673   |
|    gen/train/explained_variance     | 0.815    |
|    gen/train/learning_rate          | 0.0004   |
|    gen/train/loss                   | -0.0246  |
|    gen/train/n_updates              | 15       |
|    gen/train/policy_gradient_loss   | -0.0135  |
|    gen/train/value_loss             | 0.0463   |
--------------------------------------------------
----------------------------------------------------
| raw/                               |             |
|    gen/rollout/ep_len_mean         | 500         |
|    gen/rollout/ep_rew_mean         | 37.8        |
|    gen/rollout/ep_rew_wrapped_mean | 277         |
|    gen/time/fps                    | 6269        |
```

(continues on next page)

(continued from previous page)

```
|    gen/time/iterations             | 1           |
|    gen/time/time_elapsed           | 2           |
|    gen/time/total_timesteps        | 65536       |
|    gen/train/approx_kl             | 0.015265099 |
|    gen/train/clip_fraction         | 0.195       |
|    gen/train/clip_range            | 0.2         |
|    gen/train/entropy_loss          | -0.673      |
|    gen/train/explained_variance    | 0.815       |
|    gen/train/learning_rate         | 0.0004      |
|    gen/train/loss                  | -0.0246     |
|    gen/train/n_updates             | 15          |
|    gen/train/policy_gradient_loss  | -0.0135     |
|    gen/train/value_loss            | 0.0463      |
----------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.687    |
|    disc/disc_loss                   | 0.652    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.686    |
|    disc/disc_loss                   | 0.646    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.686    |
|    disc/disc_loss                   | 0.646    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.685    |
|    disc/disc_loss                   | 0.64     |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.685    |
|    disc/disc_loss                   | 0.638    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.684    |
|    disc/disc_loss                   | 0.634    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.682    |
|    disc/disc_loss                   | 0.628    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.682    |
|    disc/disc_loss                   | 0.625    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
```

(continues on next page)

(continued from previous page)

```
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                               |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.685    |
|    disc/disc_loss                   | 0.639    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
|    gen/rollout/ep_len_mean          | 500      |
|    gen/rollout/ep_rew_mean          | 37.8     |
|    gen/rollout/ep_rew_wrapped_mean  | 277      |
|    gen/time/fps                     | 6.27e+03 |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 2        |
|    gen/time/total_timesteps         | 6.55e+04 |
|    gen/train/approx_kl              | 0.0161   |
|    gen/train/clip_fraction          | 0.215    |
|    gen/train/clip_range             | 0.2      |
|    gen/train/entropy_loss           | -0.654   |
|    gen/train/explained_variance     | 0.892    |
|    gen/train/learning_rate          | 0.0004   |
|    gen/train/loss                   | -0.0168  |
|    gen/train/n_updates              | 20       |
|    gen/train/policy_gradient_loss   | -0.0195  |
|    gen/train/value_loss             | 0.0173   |
--------------------------------------------------
----------------------------------------------------
| raw/                               |             |
|    gen/rollout/ep_len_mean         | 500         |
|    gen/rollout/ep_rew_mean         | 40.4        |
|    gen/rollout/ep_rew_wrapped_mean | 284         |
|    gen/time/fps                    | 6275        |
|    gen/time/iterations             | 1           |
|    gen/time/time_elapsed           | 2           |
|    gen/time/total_timesteps        | 81920       |
|    gen/train/approx_kl             | 0.016116062 |
|    gen/train/clip_fraction         | 0.215       |
|    gen/train/clip_range            | 0.2         |
|    gen/train/entropy_loss          | -0.654      |
|    gen/train/explained_variance    | 0.892       |
|    gen/train/learning_rate         | 0.0004      |
|    gen/train/loss                  | -0.0168     |
|    gen/train/n_updates             | 20          |
|    gen/train/policy_gradient_loss  | -0.0195     |
|    gen/train/value_loss            | 0.0173      |
----------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.689    |
|    disc/disc_loss                   | 0.659    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.689    |
|    disc/disc_loss                   | 0.659    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.688    |
|    disc/disc_loss                   | 0.655    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0        |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.687    |
|    disc/disc_loss                   | 0.651    |
|    disc/disc_proportion_expert_pred | 0        |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.5      |
|    disc/disc_acc_expert             | 0.000977 |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.688    |
|    disc/disc_loss                   | 0.652    |
|    disc/disc_proportion_expert_pred | 0.000488 |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.573    |
|    disc/disc_acc_expert             | 0.146    |
|    disc/disc_acc_gen                | 1        |
|    disc/disc_entropy                | 0.687    |
|    disc/disc_loss                   | 0.647    |
|    disc/disc_proportion_expert_pred | 0.0728   |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.684    |
|    disc/disc_acc_expert             | 0.374    |
|    disc/disc_acc_gen                | 0.993    |
|    disc/disc_entropy                | 0.686    |
|    disc/disc_loss                   | 0.647    |
|    disc/disc_proportion_expert_pred | 0.19     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.708    |
|    disc/disc_acc_expert             | 0.434    |
|    disc/disc_acc_gen                | 0.983    |
|    disc/disc_entropy                | 0.686    |
|    disc/disc_loss                   | 0.642    |
|    disc/disc_proportion_expert_pred | 0.225    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                               |          |
|    disc/disc_acc                    | 0.558    |
|    disc/disc_acc_expert             | 0.119    |
|    disc/disc_acc_gen                | 0.997    |
|    disc/disc_entropy                | 0.687    |
|    disc/disc_loss                   | 0.652    |
|    disc/disc_proportion_expert_pred | 0.0611   |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
|    gen/rollout/ep_len_mean          | 500      |
```

(continues on next page)

(continued from previous page)

```
|    gen/rollout/ep_rew_mean          | 40.4     |
|    gen/rollout/ep_rew_wrapped_mean  | 284      |
|    gen/time/fps                     | 6.28e+03 |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 2        |
|    gen/time/total_timesteps         | 8.19e+04 |
|    gen/train/approx_kl              | 0.0112   |
|    gen/train/clip_fraction          | 0.129    |
|    gen/train/clip_range             | 0.2      |
|    gen/train/entropy_loss           | -0.634   |
|    gen/train/explained_variance     | 0.871    |
|    gen/train/learning_rate          | 0.0004   |
|    gen/train/loss                   | 0.00102  |
|    gen/train/n_updates              | 25       |
|    gen/train/policy_gradient_loss   | -0.00957 |
|    gen/train/value_loss             | 0.0103   |
--------------------------------------------------
---------------------------------------------------
| raw/                               |            |
|    gen/rollout/ep_len_mean         | 500        |
|    gen/rollout/ep_rew_mean         | 40.8       |
|    gen/rollout/ep_rew_wrapped_mean | 288        |
|    gen/time/fps                    | 6278       |
|    gen/time/iterations             | 1          |
|    gen/time/time_elapsed           | 2          |
|    gen/time/total_timesteps        | 98304      |
|    gen/train/approx_kl             | 0.01118237 |
|    gen/train/clip_fraction         | 0.129      |
|    gen/train/clip_range            | 0.2        |
|    gen/train/entropy_loss          | -0.634     |
|    gen/train/explained_variance    | 0.871      |
|    gen/train/learning_rate         | 0.0004     |
|    gen/train/loss                  | 0.00102    |
|    gen/train/n_updates             | 25         |
|    gen/train/policy_gradient_loss  | -0.00957   |
|    gen/train/value_loss            | 0.0103     |
---------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.732    |
|    disc/disc_acc_expert             | 0.468    |
|    disc/disc_acc_gen                | 0.997    |
|    disc/disc_entropy                | 0.687    |
|    disc/disc_loss                   | 0.639    |
|    disc/disc_proportion_expert_pred | 0.235    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.718    |
|    disc/disc_acc_expert             | 0.442    |
|    disc/disc_acc_gen                | 0.994    |
|    disc/disc_entropy                | 0.687    |
|    disc/disc_loss                   | 0.637    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_proportion_expert_pred | 0.224    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.736    |
|    disc/disc_acc_expert             | 0.476    |
|    disc/disc_acc_gen                | 0.996    |
|    disc/disc_entropy                | 0.687    |
|    disc/disc_loss                   | 0.638    |
|    disc/disc_proportion_expert_pred | 0.24     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.734    |
|    disc/disc_acc_expert             | 0.472    |
|    disc/disc_acc_gen                | 0.996    |
|    disc/disc_entropy                | 0.687    |
|    disc/disc_loss                   | 0.635    |
|    disc/disc_proportion_expert_pred | 0.238    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.714    |
|    disc/disc_acc_expert             | 0.44     |
|    disc/disc_acc_gen                | 0.987    |
|    disc/disc_entropy                | 0.686    |
|    disc/disc_loss                   | 0.633    |
|    disc/disc_proportion_expert_pred | 0.227    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.746    |
|    disc/disc_acc_expert             | 0.504    |
|    disc/disc_acc_gen                | 0.988    |
|    disc/disc_entropy                | 0.686    |
|    disc/disc_loss                   | 0.632    |
|    disc/disc_proportion_expert_pred | 0.258    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
```

(continues on next page)

(continued from previous page)

```
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.819    |
|    disc/disc_acc_expert             | 0.657    |
|    disc/disc_acc_gen                | 0.981    |
|    disc/disc_entropy                | 0.686    |
|    disc/disc_loss                   | 0.631    |
|    disc/disc_proportion_expert_pred | 0.338    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.856    |
|    disc/disc_acc_expert             | 0.733    |
|    disc/disc_acc_gen                | 0.979    |
|    disc/disc_entropy                | 0.685    |
|    disc/disc_loss                   | 0.627    |
|    disc/disc_proportion_expert_pred | 0.377    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                               |          |
|    disc/disc_acc                    | 0.757    |
|    disc/disc_acc_expert             | 0.524    |
|    disc/disc_acc_gen                | 0.99     |
|    disc/disc_entropy                | 0.687    |
|    disc/disc_loss                   | 0.634    |
|    disc/disc_proportion_expert_pred | 0.267    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
|    gen/rollout/ep_len_mean          | 500      |
|    gen/rollout/ep_rew_mean          | 40.8     |
|    gen/rollout/ep_rew_wrapped_mean  | 288      |
|    gen/time/fps                     | 6.28e+03 |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 2        |
|    gen/time/total_timesteps         | 9.83e+04 |
|    gen/train/approx_kl              | 0.00629  |
|    gen/train/clip_fraction          | 0.0466   |
|    gen/train/clip_range             | 0.2      |
|    gen/train/entropy_loss           | -0.635   |
|    gen/train/explained_variance     | 0.873    |
|    gen/train/learning_rate          | 0.0004   |
|    gen/train/loss                   | 0.0116   |
|    gen/train/n_updates              | 30       |
|    gen/train/policy_gradient_loss   | -0.00363 |
|    gen/train/value_loss             | 0.0126   |
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
-----------------------------------------------------
| raw/                               |              |
|    gen/rollout/ep_len_mean         | 500          |
|    gen/rollout/ep_rew_mean         | 39.6         |
|    gen/rollout/ep_rew_wrapped_mean | 287          |
|    gen/time/fps                    | 6297         |
|    gen/time/iterations             | 1            |
|    gen/time/time_elapsed           | 2            |
|    gen/time/total_timesteps        | 114688       |
|    gen/train/approx_kl             | 0.0062911767 |
|    gen/train/clip_fraction         | 0.0466       |
|    gen/train/clip_range            | 0.2          |
|    gen/train/entropy_loss          | -0.635       |
|    gen/train/explained_variance    | 0.873        |
|    gen/train/learning_rate         | 0.0004       |
|    gen/train/loss                  | 0.0116       |
|    gen/train/n_updates             | 30           |
|    gen/train/policy_gradient_loss  | -0.00363     |
|    gen/train/value_loss            | 0.0126       |
-----------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.852    |
|    disc/disc_acc_expert             | 0.735    |
|    disc/disc_acc_gen                | 0.969    |
|    disc/disc_entropy                | 0.683    |
|    disc/disc_loss                   | 0.62     |
|    disc/disc_proportion_expert_pred | 0.383    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 7        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.89     |
|    disc/disc_acc_expert             | 0.812    |
|    disc/disc_acc_gen                | 0.968    |
|    disc/disc_entropy                | 0.682    |
|    disc/disc_loss                   | 0.616    |
|    disc/disc_proportion_expert_pred | 0.422    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 7        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.919    |
|    disc/disc_acc_expert             | 0.875    |
|    disc/disc_acc_gen                | 0.964    |
|    disc/disc_entropy                | 0.681    |
|    disc/disc_loss                   | 0.614    |
|    disc/disc_proportion_expert_pred | 0.456    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 7        |
|    disc/n_expert                    | 1.02e+03 |
```

(continues on next page)

(continued from previous page)

```
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.95     |
|    disc/disc_acc_expert             | 0.933    |
|    disc/disc_acc_gen                | 0.967    |
|    disc/disc_entropy                | 0.681    |
|    disc/disc_loss                   | 0.61     |
|    disc/disc_proportion_expert_pred | 0.483    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 7        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.963    |
|    disc/disc_acc_expert             | 0.955    |
|    disc/disc_acc_gen                | 0.971    |
|    disc/disc_entropy                | 0.681    |
|    disc/disc_loss                   | 0.608    |
|    disc/disc_proportion_expert_pred | 0.492    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 7        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.962    |
|    disc/disc_acc_expert             | 0.962    |
|    disc/disc_acc_gen                | 0.963    |
|    disc/disc_entropy                | 0.679    |
|    disc/disc_loss                   | 0.603    |
|    disc/disc_proportion_expert_pred | 0.5      |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 7        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.967    |
|    disc/disc_acc_expert             | 0.97     |
|    disc/disc_acc_gen                | 0.964    |
|    disc/disc_entropy                | 0.679    |
|    disc/disc_loss                   | 0.602    |
|    disc/disc_proportion_expert_pred | 0.503    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 7        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.958    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_acc_expert             | 0.977    |
|    disc/disc_acc_gen                | 0.94     |
|    disc/disc_entropy                | 0.678    |
|    disc/disc_loss                   | 0.597    |
|    disc/disc_proportion_expert_pred | 0.518    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 7        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                               |          |
|    disc/disc_acc                    | 0.933    |
|    disc/disc_acc_expert             | 0.902    |
|    disc/disc_acc_gen                | 0.963    |
|    disc/disc_entropy                | 0.681    |
|    disc/disc_loss                   | 0.609    |
|    disc/disc_proportion_expert_pred | 0.47     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 7        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
|    gen/rollout/ep_len_mean          | 500      |
|    gen/rollout/ep_rew_mean          | 39.6     |
|    gen/rollout/ep_rew_wrapped_mean  | 287      |
|    gen/time/fps                     | 6.3e+03  |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 2        |
|    gen/time/total_timesteps         | 1.15e+05 |
|    gen/train/approx_kl              | 0.0087   |
|    gen/train/clip_fraction          | 0.0778   |
|    gen/train/clip_range             | 0.2      |
|    gen/train/entropy_loss           | -0.629   |
|    gen/train/explained_variance     | 0.928    |
|    gen/train/learning_rate          | 0.0004   |
|    gen/train/loss                   | 0.0141   |
|    gen/train/n_updates              | 35       |
|    gen/train/policy_gradient_loss   | -0.00673 |
|    gen/train/value_loss             | 0.0171   |
--------------------------------------------------
----------------------------------------------------
| raw/                               |             |
|    gen/rollout/ep_len_mean         | 500         |
|    gen/rollout/ep_rew_mean         | 39          |
|    gen/rollout/ep_rew_wrapped_mean | 282         |
|    gen/time/fps                    | 6279        |
|    gen/time/iterations             | 1           |
|    gen/time/time_elapsed           | 2           |
|    gen/time/total_timesteps        | 131072      |
|    gen/train/approx_kl             | 0.008696594 |
|    gen/train/clip_fraction         | 0.0778      |
|    gen/train/clip_range            | 0.2         |
|    gen/train/entropy_loss          | -0.629      |
|    gen/train/explained_variance    | 0.928       |
|    gen/train/learning_rate         | 0.0004      |
|    gen/train/loss                  | 0.0141      |
|    gen/train/n_updates             | 35          |
```

(continues on next page)

(continued from previous page)

```
|    gen/train/policy_gradient_loss  | -0.00673   |
|    gen/train/value_loss            | 0.0171     |
----------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.964    |
|    disc/disc_acc_expert            | 0.981    |
|    disc/disc_acc_gen               | 0.946    |
|    disc/disc_entropy               | 0.671    |
|    disc/disc_loss                  | 0.572    |
|    disc/disc_proportion_expert_pred | 0.518    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 8        |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.977    |
|    disc/disc_acc_expert            | 0.992    |
|    disc/disc_acc_gen               | 0.962    |
|    disc/disc_entropy               | 0.669    |
|    disc/disc_loss                  | 0.563    |
|    disc/disc_proportion_expert_pred | 0.515    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 8        |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.969    |
|    disc/disc_acc_expert            | 0.994    |
|    disc/disc_acc_gen               | 0.943    |
|    disc/disc_entropy               | 0.669    |
|    disc/disc_loss                  | 0.562    |
|    disc/disc_proportion_expert_pred | 0.525    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 8        |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.97     |
|    disc/disc_acc_expert            | 0.998    |
|    disc/disc_acc_gen               | 0.941    |
|    disc/disc_entropy               | 0.667    |
|    disc/disc_loss                  | 0.557    |
|    disc/disc_proportion_expert_pred | 0.528    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 8        |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_acc                    | 0.97     |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.939    |
|    disc/disc_entropy                | 0.665    |
|    disc/disc_loss                   | 0.553    |
|    disc/disc_proportion_expert_pred | 0.53     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 8        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.972    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.943    |
|    disc/disc_entropy                | 0.666    |
|    disc/disc_loss                   | 0.552    |
|    disc/disc_proportion_expert_pred | 0.528    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 8        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.969    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.938    |
|    disc/disc_entropy                | 0.663    |
|    disc/disc_loss                   | 0.543    |
|    disc/disc_proportion_expert_pred | 0.531    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 8        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.976    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.952    |
|    disc/disc_entropy                | 0.661    |
|    disc/disc_loss                   | 0.539    |
|    disc/disc_proportion_expert_pred | 0.524    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 8        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                               |          |
|    disc/disc_acc                    | 0.971    |
|    disc/disc_acc_expert             | 0.996    |
|    disc/disc_acc_gen                | 0.946    |
|    disc/disc_entropy                | 0.667    |
|    disc/disc_loss                   | 0.555    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_proportion_expert_pred | 0.525    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 8        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
|    gen/rollout/ep_len_mean          | 500      |
|    gen/rollout/ep_rew_mean          | 39       |
|    gen/rollout/ep_rew_wrapped_mean  | 282      |
|    gen/time/fps                     | 6.28e+03 |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 2        |
|    gen/time/total_timesteps         | 1.31e+05 |
|    gen/train/approx_kl              | 0.00855  |
|    gen/train/clip_fraction          | 0.0715   |
|    gen/train/clip_range             | 0.2      |
|    gen/train/entropy_loss           | -0.624   |
|    gen/train/explained_variance     | 0.922    |
|    gen/train/learning_rate          | 0.0004   |
|    gen/train/loss                   | 0.00161  |
|    gen/train/n_updates              | 40       |
|    gen/train/policy_gradient_loss   | -0.00499 |
|    gen/train/value_loss             | 0.0237   |
--------------------------------------------------
----------------------------------------------------
| raw/                               |             |
|    gen/rollout/ep_len_mean         | 500         |
|    gen/rollout/ep_rew_mean         | 39.6        |
|    gen/rollout/ep_rew_wrapped_mean | 271         |
|    gen/time/fps                    | 6297        |
|    gen/time/iterations             | 1           |
|    gen/time/time_elapsed           | 2           |
|    gen/time/total_timesteps        | 147456      |
|    gen/train/approx_kl             | 0.008551636 |
|    gen/train/clip_fraction         | 0.0715      |
|    gen/train/clip_range            | 0.2         |
|    gen/train/entropy_loss          | -0.624      |
|    gen/train/explained_variance    | 0.922       |
|    gen/train/learning_rate         | 0.0004      |
|    gen/train/loss                  | 0.00161     |
|    gen/train/n_updates             | 40          |
|    gen/train/policy_gradient_loss  | -0.00499    |
|    gen/train/value_loss            | 0.0237      |
----------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.96     |
|    disc/disc_acc_expert             | 0.998    |
|    disc/disc_acc_gen                | 0.922    |
|    disc/disc_entropy                | 0.674    |
|    disc/disc_loss                   | 0.571    |
|    disc/disc_proportion_expert_pred | 0.538    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 9        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
| raw/                                |          |
|    disc/disc_acc                    | 0.956    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.912    |
|    disc/disc_entropy                | 0.672    |
|    disc/disc_loss                   | 0.567    |
|    disc/disc_proportion_expert_pred | 0.544    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 9        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.962    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.924    |
|    disc/disc_entropy                | 0.67     |
|    disc/disc_loss                   | 0.56     |
|    disc/disc_proportion_expert_pred | 0.538    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 9        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.966    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.932    |
|    disc/disc_entropy                | 0.669    |
|    disc/disc_loss                   | 0.558    |
|    disc/disc_proportion_expert_pred | 0.534    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 9        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.954    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.907    |
|    disc/disc_entropy                | 0.667    |
|    disc/disc_loss                   | 0.553    |
|    disc/disc_proportion_expert_pred | 0.546    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 9        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.951    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.902    |
|    disc/disc_entropy                | 0.667    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_loss                   | 0.551    |
|    disc/disc_proportion_expert_pred | 0.549    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 9        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.96     |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.921    |
|    disc/disc_entropy                | 0.662    |
|    disc/disc_loss                   | 0.539    |
|    disc/disc_proportion_expert_pred | 0.54     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 9        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.958    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.915    |
|    disc/disc_entropy                | 0.661    |
|    disc/disc_loss                   | 0.535    |
|    disc/disc_proportion_expert_pred | 0.542    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 9        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                               |          |
|    disc/disc_acc                    | 0.958    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.917    |
|    disc/disc_entropy                | 0.668    |
|    disc/disc_loss                   | 0.554    |
|    disc/disc_proportion_expert_pred | 0.541    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 9        |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
|    gen/rollout/ep_len_mean          | 500      |
|    gen/rollout/ep_rew_mean          | 39.6     |
|    gen/rollout/ep_rew_wrapped_mean  | 271      |
|    gen/time/fps                     | 6.3e+03  |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 2        |
|    gen/time/total_timesteps         | 1.47e+05 |
|    gen/train/approx_kl              | 0.00591  |
|    gen/train/clip_fraction          | 0.0515   |
|    gen/train/clip_range             | 0.2      |
|    gen/train/entropy_loss           | -0.613   |
|    gen/train/explained_variance     | 0.935    |
```

(continues on next page)

(continued from previous page)

```
|    gen/train/learning_rate          | 0.0004   |
|    gen/train/loss                   | -0.00763 |
|    gen/train/n_updates              | 45       |
|    gen/train/policy_gradient_loss   | -0.00313 |
|    gen/train/value_loss             | 0.0288   |
---------------------------------------------------
------------------------------------------------------
| raw/                               |              |
|    gen/rollout/ep_len_mean         | 500          |
|    gen/rollout/ep_rew_mean         | 44.7         |
|    gen/rollout/ep_rew_wrapped_mean | 259          |
|    gen/time/fps                    | 6284         |
|    gen/time/iterations             | 1            |
|    gen/time/time_elapsed           | 2            |
|    gen/time/total_timesteps        | 163840       |
|    gen/train/approx_kl             | 0.0059148837 |
|    gen/train/clip_fraction         | 0.0515       |
|    gen/train/clip_range            | 0.2          |
|    gen/train/entropy_loss          | -0.613       |
|    gen/train/explained_variance    | 0.935        |
|    gen/train/learning_rate         | 0.0004       |
|    gen/train/loss                  | -0.00763     |
|    gen/train/n_updates             | 45           |
|    gen/train/policy_gradient_loss  | -0.00313     |
|    gen/train/value_loss            | 0.0288       |
------------------------------------------------------
---------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.951    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.901    |
|    disc/disc_entropy                | 0.643    |
|    disc/disc_loss                   | 0.495    |
|    disc/disc_proportion_expert_pred | 0.549    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 10       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
---------------------------------------------------
---------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.948    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.896    |
|    disc/disc_entropy                | 0.638    |
|    disc/disc_loss                   | 0.485    |
|    disc/disc_proportion_expert_pred | 0.552    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 10       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
---------------------------------------------------
---------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.955    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.909    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_entropy               | 0.636    |
|    disc/disc_loss                  | 0.481    |
|    disc/disc_proportion_expert_pred | 0.545    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 10       |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.95     |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.899    |
|    disc/disc_entropy               | 0.633    |
|    disc/disc_loss                  | 0.477    |
|    disc/disc_proportion_expert_pred | 0.55     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 10       |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.945    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.89     |
|    disc/disc_entropy               | 0.633    |
|    disc/disc_loss                  | 0.475    |
|    disc/disc_proportion_expert_pred | 0.555    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 10       |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.942    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.885    |
|    disc/disc_entropy               | 0.628    |
|    disc/disc_loss                  | 0.468    |
|    disc/disc_proportion_expert_pred | 0.558    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 10       |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.948    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.896    |
|    disc/disc_entropy               | 0.624    |
|    disc/disc_loss                  | 0.461    |
|    disc/disc_proportion_expert_pred | 0.552    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 10       |
```

(continues on next page)

(continued from previous page)

```
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.95     |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.899    |
|    disc/disc_entropy                | 0.615    |
|    disc/disc_loss                   | 0.446    |
|    disc/disc_proportion_expert_pred | 0.55     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 10       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                               |          |
|    disc/disc_acc                    | 0.948    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.897    |
|    disc/disc_entropy                | 0.631    |
|    disc/disc_loss                   | 0.474    |
|    disc/disc_proportion_expert_pred | 0.552    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 10       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
|    gen/rollout/ep_len_mean          | 500      |
|    gen/rollout/ep_rew_mean          | 44.7     |
|    gen/rollout/ep_rew_wrapped_mean  | 259      |
|    gen/time/fps                     | 6.28e+03 |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 2        |
|    gen/time/total_timesteps         | 1.64e+05 |
|    gen/train/approx_kl              | 0.00881  |
|    gen/train/clip_fraction          | 0.0822   |
|    gen/train/clip_range             | 0.2      |
|    gen/train/entropy_loss           | -0.596   |
|    gen/train/explained_variance     | 0.942    |
|    gen/train/learning_rate          | 0.0004   |
|    gen/train/loss                   | -0.0335  |
|    gen/train/n_updates              | 50       |
|    gen/train/policy_gradient_loss   | -0.00478 |
|    gen/train/value_loss             | 0.0465   |
--------------------------------------------------
-----------------------------------------------------
| raw/                               |              |
|    gen/rollout/ep_len_mean         | 500          |
|    gen/rollout/ep_rew_mean         | 50.9         |
|    gen/rollout/ep_rew_wrapped_mean | 243          |
|    gen/time/fps                    | 6427         |
|    gen/time/iterations             | 1            |
|    gen/time/time_elapsed           | 2            |
|    gen/time/total_timesteps        | 180224       |
|    gen/train/approx_kl             | 0.0088136345 |
|    gen/train/clip_fraction         | 0.0822       |
```

(continues on next page)

(continued from previous page)

```
|    gen/train/clip_range            | 0.2          |
|    gen/train/entropy_loss          | -0.596       |
|    gen/train/explained_variance    | 0.942        |
|    gen/train/learning_rate         | 0.0004       |
|    gen/train/loss                  | -0.0335      |
|    gen/train/n_updates             | 50           |
|    gen/train/policy_gradient_loss  | -0.00478     |
|    gen/train/value_loss            | 0.0465       |
-----------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.788    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.575    |
|    disc/disc_entropy                | 0.642    |
|    disc/disc_loss                   | 0.543    |
|    disc/disc_proportion_expert_pred | 0.712    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 11       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.791    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.581    |
|    disc/disc_entropy                | 0.637    |
|    disc/disc_loss                   | 0.536    |
|    disc/disc_proportion_expert_pred | 0.709    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 11       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.794    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.588    |
|    disc/disc_entropy                | 0.632    |
|    disc/disc_loss                   | 0.526    |
|    disc/disc_proportion_expert_pred | 0.706    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 11       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.781    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.562    |
|    disc/disc_entropy                | 0.632    |
|    disc/disc_loss                   | 0.533    |
|    disc/disc_proportion_expert_pred | 0.719    |
|    disc/disc_proportion_expert_true | 0.5      |
```

(continues on next page)

(continued from previous page)

```
|    disc/global_step                 | 11       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.788    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.575    |
|    disc/disc_entropy                | 0.628    |
|    disc/disc_loss                   | 0.524    |
|    disc/disc_proportion_expert_pred | 0.712    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 11       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.783    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.566    |
|    disc/disc_entropy                | 0.624    |
|    disc/disc_loss                   | 0.524    |
|    disc/disc_proportion_expert_pred | 0.717    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 11       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.787    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.573    |
|    disc/disc_entropy                | 0.622    |
|    disc/disc_loss                   | 0.519    |
|    disc/disc_proportion_expert_pred | 0.713    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 11       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.775    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.551    |
|    disc/disc_entropy                | 0.621    |
|    disc/disc_loss                   | 0.524    |
|    disc/disc_proportion_expert_pred | 0.725    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 11       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
| mean/                              |          |
|    disc/disc_acc                   | 0.786    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.572    |
|    disc/disc_entropy               | 0.63     |
|    disc/disc_loss                  | 0.529    |
|    disc/disc_proportion_expert_pred | 0.714    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 11       |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
|    gen/rollout/ep_len_mean         | 500      |
|    gen/rollout/ep_rew_mean         | 50.9     |
|    gen/rollout/ep_rew_wrapped_mean  | 243      |
|    gen/time/fps                    | 6.43e+03 |
|    gen/time/iterations             | 1        |
|    gen/time/time_elapsed           | 2        |
|    gen/time/total_timesteps        | 1.8e+05  |
|    gen/train/approx_kl             | 0.00988  |
|    gen/train/clip_fraction         | 0.117    |
|    gen/train/clip_range            | 0.2      |
|    gen/train/entropy_loss          | -0.597   |
|    gen/train/explained_variance    | 0.95     |
|    gen/train/learning_rate         | 0.0004   |
|    gen/train/loss                  | 0.0114   |
|    gen/train/n_updates             | 55       |
|    gen/train/policy_gradient_loss  | -0.00606 |
|    gen/train/value_loss            | 0.0522   |
--------------------------------------------------
----------------------------------------------------
| raw/                              |             |
|    gen/rollout/ep_len_mean        | 500         |
|    gen/rollout/ep_rew_mean        | 56.4        |
|    gen/rollout/ep_rew_wrapped_mean | 229         |
|    gen/time/fps                   | 6408        |
|    gen/time/iterations            | 1           |
|    gen/time/time_elapsed          | 2           |
|    gen/time/total_timesteps       | 196608      |
|    gen/train/approx_kl            | 0.009878516 |
|    gen/train/clip_fraction        | 0.117       |
|    gen/train/clip_range           | 0.2         |
|    gen/train/entropy_loss         | -0.597      |
|    gen/train/explained_variance   | 0.95        |
|    gen/train/learning_rate        | 0.0004      |
|    gen/train/loss                 | 0.0114      |
|    gen/train/n_updates            | 55          |
|    gen/train/policy_gradient_loss | -0.00606    |
|    gen/train/value_loss           | 0.0522      |
----------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.583    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.165    |
|    disc/disc_entropy               | 0.671    |
|    disc/disc_loss                  | 0.659    |
|    disc/disc_proportion_expert_pred | 0.917    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 12       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.588    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.177    |
|    disc/disc_entropy                | 0.672    |
|    disc/disc_loss                   | 0.653    |
|    disc/disc_proportion_expert_pred | 0.912    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 12       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.588    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.176    |
|    disc/disc_entropy                | 0.671    |
|    disc/disc_loss                   | 0.653    |
|    disc/disc_proportion_expert_pred | 0.912    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 12       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.582    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.163    |
|    disc/disc_entropy                | 0.672    |
|    disc/disc_loss                   | 0.653    |
|    disc/disc_proportion_expert_pred | 0.918    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 12       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.596    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.192    |
|    disc/disc_entropy                | 0.673    |
|    disc/disc_loss                   | 0.65     |
|    disc/disc_proportion_expert_pred | 0.904    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 12       |
|    disc/n_expert                    | 1.02e+03 |
|    disc/n_generated                 | 1.02e+03 |
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.596    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.192    |
|    disc/disc_entropy               | 0.674    |
|    disc/disc_loss                  | 0.646    |
|    disc/disc_proportion_expert_pred | 0.904    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 12       |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.61     |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.221    |
|    disc/disc_entropy               | 0.676    |
|    disc/disc_loss                  | 0.645    |
|    disc/disc_proportion_expert_pred | 0.89     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 12       |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.604    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.208    |
|    disc/disc_entropy               | 0.675    |
|    disc/disc_loss                  | 0.643    |
|    disc/disc_proportion_expert_pred | 0.896    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 12       |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                              |          |
|    disc/disc_acc                   | 0.593    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.187    |
|    disc/disc_entropy               | 0.673    |
|    disc/disc_loss                  | 0.65     |
|    disc/disc_proportion_expert_pred | 0.907    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 12       |
|    disc/n_expert                   | 1.02e+03 |
|    disc/n_generated                | 1.02e+03 |
|    gen/rollout/ep_len_mean         | 500      |
|    gen/rollout/ep_rew_mean         | 56.4     |
|    gen/rollout/ep_rew_wrapped_mean | 229      |
|    gen/time/fps                    | 6.41e+03 |
|    gen/time/iterations             | 1        |
|    gen/time/time_elapsed           | 2        |
```

(continues on next page)

(continued from previous page)

```
|    gen/time/total_timesteps          | 1.97e+05 |
|    gen/train/approx_kl               | 0.0124   |
|    gen/train/clip_fraction           | 0.148    |
|    gen/train/clip_range              | 0.2      |
|    gen/train/entropy_loss            | -0.586   |
|    gen/train/explained_variance      | 0.968    |
|    gen/train/learning_rate           | 0.0004   |
|    gen/train/loss                    | 0.000133 |
|    gen/train/n_updates               | 60       |
|    gen/train/policy_gradient_loss    | -0.00875 |
|    gen/train/value_loss              | 0.0555   |
--------------------------------------------------
```

… and finally evaluate it again.

```
env.seed(SEED)
learner_rewards_after_training, _ = evaluate_policy(
    learner, env, 100, return_episode_rewards=True
)
```

We can see that an untrained policy performs poorly, while GAIL matches expert returns (500):

```
print(
    "Rewards before training:",
    np.mean(learner_rewards_before_training),
    "+/-",
    np.std(learner_rewards_before_training),
)
print(
    "Rewards after training:",
    np.mean(learner_rewards_after_training),
    "+/-",
    np.std(learner_rewards_after_training),
)
```

```
Rewards before training: 102.6 +/- 24.11514047232568
Rewards after training: 49.76 +/- 16.98535840069323
```

download this notebook here

## 2.22 Train an Agent using Adversarial Inverse Reinforcement Learning

As usual, we first need an expert. Again, we download one from the HuggingFace model hub for convenience.

Note that we now use a variant of the CartPole environment from the seals package, which has fixed episode durations. Read more about why we do this here.

```
import numpy as np
from imitation.policies.serialize import load_policy
from imitation.util.util import make_vec_env
from imitation.data.wrappers import RolloutInfoWrapper

```

(continues on next page)

(continued from previous page)

```
SEED = 42

FAST = True

if FAST:
    N_RL_TRAIN_STEPS = 100_000
else:
    N_RL_TRAIN_STEPS = 2_000_000

venv = make_vec_env(
    "seals:seals/CartPole-v0",
    rng=np.random.default_rng(SEED),
    n_envs=8,
    post_wrappers=[
        lambda env, _: RolloutInfoWrapper(env)
    ],  # needed for computing rollouts later
)
expert = load_policy(
    "ppo-huggingface",
    organization="HumanCompatibleAI",
    env_name="seals/CartPole-v0",
    venv=venv,
)
```

We generate some expert trajectories, that the discriminator needs to distinguish from the learner's trajectories.

```
from imitation.data import rollout

rollouts = rollout.rollout(
    expert,
    venv,
    rollout.make_sample_until(min_timesteps=None, min_episodes=60),
    rng=np.random.default_rng(SEED),
)
```

Now we are ready to set up our AIRL trainer. Note, that the `reward_net` is actually the network of the discriminator. We evaluate the learner before and after training so we can see if it made any progress.

```
from imitation.algorithms.adversarial.airl import AIRL
from imitation.rewards.reward_nets import BasicShapedRewardNet
from imitation.util.networks import RunningNorm
from stable_baselines3 import PPO
from stable_baselines3.ppo import MlpPolicy
from stable_baselines3.common.evaluation import evaluate_policy


learner = PPO(
    env=venv,
    policy=MlpPolicy,
    batch_size=64,
    ent_coef=0.0,
    learning_rate=0.0005,
    gamma=0.95,
    clip_range=0.1,
    vf_coef=0.1,
    n_epochs=5,
```

(continues on next page)

(continued from previous page)

```
    seed=SEED,
)
reward_net = BasicShapedRewardNet(
    observation_space=venv.observation_space,
    action_space=venv.action_space,
    normalize_input_layer=RunningNorm,
)
airl_trainer = AIRL(
    demonstrations=rollouts,
    demo_batch_size=2048,
    gen_replay_buffer_capacity=512,
    n_disc_updates_per_round=16,
    venv=venv,
    gen_algo=learner,
    reward_net=reward_net,
)

venv.seed(SEED)
learner_rewards_before_training, _ = evaluate_policy(
    learner, venv, 100, return_episode_rewards=True
)
airl_trainer.train(N_RL_TRAIN_STEPS)
venv.seed(SEED)
learner_rewards_after_training, _ = evaluate_policy(
    learner, venv, 100, return_episode_rewards=True
)
```

```
------------------------------------------
| raw/                        |           |
|    gen/rollout/ep_len_mean  | 500       |
|    gen/rollout/ep_rew_mean  | 33.1      |
|    gen/time/fps             | 5119      |
|    gen/time/iterations      | 1         |
|    gen/time/time_elapsed    | 3         |
|    gen/time/total_timesteps | 16384     |
------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.581    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.162    |
|    disc/disc_entropy                | 0.664    |
|    disc/disc_loss                   | 0.676    |
|    disc/disc_proportion_expert_pred | 0.919    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.586    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.172    |
|    disc/disc_entropy                | 0.664    |
|    disc/disc_loss                   | 0.673    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_proportion_expert_pred | 0.914    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.593    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.186    |
|    disc/disc_entropy                | 0.665    |
|    disc/disc_loss                   | 0.669    |
|    disc/disc_proportion_expert_pred | 0.907    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.591    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.182    |
|    disc/disc_entropy                | 0.666    |
|    disc/disc_loss                   | 0.672    |
|    disc/disc_proportion_expert_pred | 0.909    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.598    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.197    |
|    disc/disc_entropy                | 0.666    |
|    disc/disc_loss                   | 0.665    |
|    disc/disc_proportion_expert_pred | 0.902    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.606    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.211    |
|    disc/disc_entropy                | 0.666    |
|    disc/disc_loss                   | 0.662    |
|    disc/disc_proportion_expert_pred | 0.894    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
```

(continues on next page)

(continued from previous page)

```
--------------------------------------------------
--------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.605    |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.21     |
|    disc/disc_entropy                 | 0.667    |
|    disc/disc_loss                    | 0.659    |
|    disc/disc_proportion_expert_pred  | 0.895    |
|    disc/disc_proportion_expert_true  | 0.5      |
|    disc/global_step                  | 1        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.598    |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.196    |
|    disc/disc_entropy                 | 0.667    |
|    disc/disc_loss                    | 0.66     |
|    disc/disc_proportion_expert_pred  | 0.902    |
|    disc/disc_proportion_expert_true  | 0.5      |
|    disc/global_step                  | 1        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.613    |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.226    |
|    disc/disc_entropy                 | 0.668    |
|    disc/disc_loss                    | 0.654    |
|    disc/disc_proportion_expert_pred  | 0.887    |
|    disc/disc_proportion_expert_true  | 0.5      |
|    disc/global_step                  | 1        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.623    |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.246    |
|    disc/disc_entropy                 | 0.668    |
|    disc/disc_loss                    | 0.65     |
|    disc/disc_proportion_expert_pred  | 0.877    |
|    disc/disc_proportion_expert_true  | 0.5      |
|    disc/global_step                  | 1        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.617    |
|    disc/disc_acc_expert              | 1        |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_acc_gen                | 0.235    |
|    disc/disc_entropy                | 0.668    |
|    disc/disc_loss                   | 0.651    |
|    disc/disc_proportion_expert_pred | 0.883    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.632    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.264    |
|    disc/disc_entropy                | 0.668    |
|    disc/disc_loss                   | 0.645    |
|    disc/disc_proportion_expert_pred | 0.868    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.629    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.258    |
|    disc/disc_entropy                | 0.668    |
|    disc/disc_loss                   | 0.644    |
|    disc/disc_proportion_expert_pred | 0.871    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.643    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.286    |
|    disc/disc_entropy                | 0.669    |
|    disc/disc_loss                   | 0.641    |
|    disc/disc_proportion_expert_pred | 0.857    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.646    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.292    |
|    disc/disc_entropy                | 0.669    |
|    disc/disc_loss                   | 0.637    |
|    disc/disc_proportion_expert_pred | 0.854    |
|    disc/disc_proportion_expert_true | 0.5      |
```

(continues on next page)

(continued from previous page)

```
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.653    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.305    |
|    disc/disc_entropy                | 0.668    |
|    disc/disc_loss                   | 0.633    |
|    disc/disc_proportion_expert_pred | 0.847    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| mean/                               |          |
|    disc/disc_acc                    | 0.613    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.227    |
|    disc/disc_entropy                | 0.667    |
|    disc/disc_loss                   | 0.656    |
|    disc/disc_proportion_expert_pred | 0.887    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 1        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
|    gen/rollout/ep_len_mean          | 500      |
|    gen/rollout/ep_rew_mean          | 33.1     |
|    gen/time/fps                     | 5.12e+03 |
|    gen/time/iterations              | 1        |
|    gen/time/time_elapsed            | 3        |
|    gen/time/total_timesteps         | 1.64e+04 |
|    gen/train/approx_kl              | 0.00136  |
|    gen/train/clip_fraction          | 0.0238   |
|    gen/train/clip_range             | 0.1      |
|    gen/train/entropy_loss           | -0.692   |
|    gen/train/explained_variance     | -0.0116  |
|    gen/train/learning_rate          | 0.0005   |
|    gen/train/loss                   | 3.17     |
|    gen/train/n_updates              | 5        |
|    gen/train/policy_gradient_loss   | 7.75e-06 |
|    gen/train/value_loss             | 117      |
--------------------------------------------------
-----------------------------------------------------
| raw/                               |              |
|    gen/rollout/ep_len_mean         | 500          |
|    gen/rollout/ep_rew_mean         | 34.6         |
|    gen/rollout/ep_rew_wrapped_mean | -525         |
|    gen/time/fps                    | 5094         |
|    gen/time/iterations             | 1            |
|    gen/time/time_elapsed           | 3            |
|    gen/time/total_timesteps        | 32768        |
|    gen/train/approx_kl             | 0.0013636536 |
|    gen/train/clip_fraction         | 0.0238       |
```

(continues on next page)

(continued from previous page)

```
|    gen/train/clip_range           | 0.1          |
|    gen/train/entropy_loss         | -0.692       |
|    gen/train/explained_variance   | -0.0116      |
|    gen/train/learning_rate        | 0.0005       |
|    gen/train/loss                 | 3.17         |
|    gen/train/n_updates            | 5            |
|    gen/train/policy_gradient_loss | 7.75e-06     |
|    gen/train/value_loss           | 117          |
------------------------------------------------------
---------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.68     |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.36     |
|    disc/disc_entropy                 | 0.664    |
|    disc/disc_loss                    | 0.618    |
|    disc/disc_proportion_expert_pred | 0.82     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                  | 2        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
---------------------------------------------------
---------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.687    |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.375    |
|    disc/disc_entropy                 | 0.664    |
|    disc/disc_loss                    | 0.615    |
|    disc/disc_proportion_expert_pred | 0.813    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                  | 2        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
---------------------------------------------------
---------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.684    |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.368    |
|    disc/disc_entropy                 | 0.665    |
|    disc/disc_loss                    | 0.615    |
|    disc/disc_proportion_expert_pred | 0.816    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                  | 2        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
---------------------------------------------------
---------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.688    |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.376    |
|    disc/disc_entropy                 | 0.666    |
|    disc/disc_loss                    | 0.617    |
|    disc/disc_proportion_expert_pred | 0.812    |
|    disc/disc_proportion_expert_true | 0.5      |
```

(continues on next page)

(continued from previous page)

```
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.687    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.373    |
|    disc/disc_entropy                | 0.667    |
|    disc/disc_loss                   | 0.616    |
|    disc/disc_proportion_expert_pred | 0.813    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.684    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.368    |
|    disc/disc_entropy                | 0.668    |
|    disc/disc_loss                   | 0.619    |
|    disc/disc_proportion_expert_pred | 0.816    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.677    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.353    |
|    disc/disc_entropy                | 0.668    |
|    disc/disc_loss                   | 0.62     |
|    disc/disc_proportion_expert_pred | 0.823    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.683    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.366    |
|    disc/disc_entropy                | 0.669    |
|    disc/disc_loss                   | 0.619    |
|    disc/disc_proportion_expert_pred | 0.817    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
| raw/                                |          |
|    disc/disc_acc                    | 0.69     |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.38     |
|    disc/disc_entropy                | 0.667    |
|    disc/disc_loss                   | 0.614    |
|    disc/disc_proportion_expert_pred | 0.81     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.69     |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.381    |
|    disc/disc_entropy                | 0.669    |
|    disc/disc_loss                   | 0.615    |
|    disc/disc_proportion_expert_pred | 0.81     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.688    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.377    |
|    disc/disc_entropy                | 0.67     |
|    disc/disc_loss                   | 0.617    |
|    disc/disc_proportion_expert_pred | 0.812    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.706    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.412    |
|    disc/disc_entropy                | 0.669    |
|    disc/disc_loss                   | 0.61     |
|    disc/disc_proportion_expert_pred | 0.794    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.697    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.395    |
|    disc/disc_entropy                | 0.67     |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_loss                   | 0.613    |
|    disc/disc_proportion_expert_pred | 0.803    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.706    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.412    |
|    disc/disc_entropy                | 0.67     |
|    disc/disc_loss                   | 0.608    |
|    disc/disc_proportion_expert_pred | 0.794    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.713    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.426    |
|    disc/disc_entropy                | 0.67     |
|    disc/disc_loss                   | 0.607    |
|    disc/disc_proportion_expert_pred | 0.787    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.705    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.409    |
|    disc/disc_entropy                | 0.669    |
|    disc/disc_loss                   | 0.605    |
|    disc/disc_proportion_expert_pred | 0.795    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 2        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
---------------------------------------------------
| mean/                               |           |
|    disc/disc_acc                    | 0.692     |
|    disc/disc_acc_expert             | 1         |
|    disc/disc_acc_gen                | 0.383     |
|    disc/disc_entropy                | 0.668     |
|    disc/disc_loss                   | 0.614     |
|    disc/disc_proportion_expert_pred | 0.808     |
|    disc/disc_proportion_expert_true | 0.5       |
|    disc/global_step                 | 2         |
|    disc/n_expert                    | 2.05e+03  |
```

(continues on next page)

(continued from previous page)

```
|    disc/n_generated                 | 2.05e+03  |
|    gen/rollout/ep_len_mean          | 500       |
|    gen/rollout/ep_rew_mean          | 34.6      |
|    gen/rollout/ep_rew_wrapped_mean  | -525      |
|    gen/time/fps                     | 5.09e+03  |
|    gen/time/iterations              | 1         |
|    gen/time/time_elapsed            | 3         |
|    gen/time/total_timesteps         | 3.28e+04  |
|    gen/train/approx_kl              | 0.0011    |
|    gen/train/clip_fraction          | 0.00289   |
|    gen/train/clip_range             | 0.1       |
|    gen/train/entropy_loss           | -0.691    |
|    gen/train/explained_variance     | 0.178     |
|    gen/train/learning_rate          | 0.0005    |
|    gen/train/loss                   | 171       |
|    gen/train/n_updates              | 10        |
|    gen/train/policy_gradient_loss   | -7.06e-06 |
|    gen/train/value_loss             | 4.82e+03  |
---------------------------------------------------
-----------------------------------------------------
| raw/                               |              |
|    gen/rollout/ep_len_mean         | 500          |
|    gen/rollout/ep_rew_mean         | 35.4         |
|    gen/rollout/ep_rew_wrapped_mean | -1.47e+03    |
|    gen/time/fps                    | 5104         |
|    gen/time/iterations             | 1            |
|    gen/time/time_elapsed           | 3            |
|    gen/time/total_timesteps        | 49152        |
|    gen/train/approx_kl             | 0.0010964434 |
|    gen/train/clip_fraction         | 0.00289      |
|    gen/train/clip_range            | 0.1          |
|    gen/train/entropy_loss          | -0.691       |
|    gen/train/explained_variance    | 0.178        |
|    gen/train/learning_rate         | 0.0005       |
|    gen/train/loss                  | 171          |
|    gen/train/n_updates             | 10           |
|    gen/train/policy_gradient_loss  | -7.06e-06    |
|    gen/train/value_loss            | 4.82e+03     |
-----------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.707    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.413    |
|    disc/disc_entropy                | 0.673    |
|    disc/disc_loss                   | 0.633    |
|    disc/disc_proportion_expert_pred | 0.793    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.711    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.422    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_entropy                | 0.673    |
|    disc/disc_loss                   | 0.631    |
|    disc/disc_proportion_expert_pred | 0.789    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.721    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.442    |
|    disc/disc_entropy                | 0.674    |
|    disc/disc_loss                   | 0.63     |
|    disc/disc_proportion_expert_pred | 0.779    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.719    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.438    |
|    disc/disc_entropy                | 0.673    |
|    disc/disc_loss                   | 0.629    |
|    disc/disc_proportion_expert_pred | 0.781    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.726    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.451    |
|    disc/disc_entropy                | 0.673    |
|    disc/disc_loss                   | 0.626    |
|    disc/disc_proportion_expert_pred | 0.774    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.729    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.458    |
|    disc/disc_entropy                | 0.674    |
|    disc/disc_loss                   | 0.623    |
|    disc/disc_proportion_expert_pred | 0.771    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
```

(continues on next page)

(continued from previous page)

```
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
---------------------------------------------------
---------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.737    |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.474    |
|    disc/disc_entropy                 | 0.674    |
|    disc/disc_loss                    | 0.62     |
|    disc/disc_proportion_expert_pred | 0.763    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                  | 3        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
---------------------------------------------------
---------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.749    |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.497    |
|    disc/disc_entropy                 | 0.675    |
|    disc/disc_loss                    | 0.615    |
|    disc/disc_proportion_expert_pred | 0.751    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                  | 3        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
---------------------------------------------------
---------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.743    |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.485    |
|    disc/disc_entropy                 | 0.674    |
|    disc/disc_loss                    | 0.618    |
|    disc/disc_proportion_expert_pred | 0.757    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                  | 3        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
---------------------------------------------------
---------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.74     |
|    disc/disc_acc_expert              | 1        |
|    disc/disc_acc_gen                 | 0.479    |
|    disc/disc_entropy                 | 0.675    |
|    disc/disc_loss                    | 0.617    |
|    disc/disc_proportion_expert_pred | 0.76     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                  | 3        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
---------------------------------------------------
---------------------------------------------------
| raw/                                 |          |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_acc                    | 0.752    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.504    |
|    disc/disc_entropy                | 0.675    |
|    disc/disc_loss                   | 0.611    |
|    disc/disc_proportion_expert_pred | 0.748    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.764    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.528    |
|    disc/disc_entropy                | 0.674    |
|    disc/disc_loss                   | 0.609    |
|    disc/disc_proportion_expert_pred | 0.736    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.758    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.516    |
|    disc/disc_entropy                | 0.674    |
|    disc/disc_loss                   | 0.61     |
|    disc/disc_proportion_expert_pred | 0.742    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.759    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.519    |
|    disc/disc_entropy                | 0.675    |
|    disc/disc_loss                   | 0.609    |
|    disc/disc_proportion_expert_pred | 0.741    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.769    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.537    |
|    disc/disc_entropy                | 0.674    |
|    disc/disc_loss                   | 0.604    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_proportion_expert_pred | 0.731    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.778    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.555    |
|    disc/disc_entropy                | 0.674    |
|    disc/disc_loss                   | 0.599    |
|    disc/disc_proportion_expert_pred | 0.722    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 3        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
---------------------------------------------------
| mean/                               |           |
|    disc/disc_acc                    | 0.741     |
|    disc/disc_acc_expert             | 1         |
|    disc/disc_acc_gen                | 0.482     |
|    disc/disc_entropy                | 0.674     |
|    disc/disc_loss                   | 0.618     |
|    disc/disc_proportion_expert_pred | 0.759     |
|    disc/disc_proportion_expert_true | 0.5       |
|    disc/global_step                 | 3         |
|    disc/n_expert                    | 2.05e+03  |
|    disc/n_generated                 | 2.05e+03  |
|    gen/rollout/ep_len_mean          | 500       |
|    gen/rollout/ep_rew_mean          | 35.4      |
|    gen/rollout/ep_rew_wrapped_mean  | -1.47e+03 |
|    gen/time/fps                     | 5.1e+03   |
|    gen/time/iterations              | 1         |
|    gen/time/time_elapsed            | 3         |
|    gen/time/total_timesteps         | 4.92e+04  |
|    gen/train/approx_kl              | 0.00162   |
|    gen/train/clip_fraction          | 0.0488    |
|    gen/train/clip_range             | 0.1       |
|    gen/train/entropy_loss           | -0.691    |
|    gen/train/explained_variance     | 0.66      |
|    gen/train/learning_rate          | 0.0005    |
|    gen/train/loss                   | 89.1      |
|    gen/train/n_updates              | 15        |
|    gen/train/policy_gradient_loss   | -0.00034  |
|    gen/train/value_loss             | 1.37e+03  |
---------------------------------------------------
-----------------------------------------------------
| raw/                               |              |
|    gen/rollout/ep_len_mean         | 500          |
|    gen/rollout/ep_rew_mean         | 38.2         |
|    gen/rollout/ep_rew_wrapped_mean | -1.52e+03    |
|    gen/time/fps                    | 5118         |
|    gen/time/iterations             | 1            |
|    gen/time/time_elapsed           | 3            |
```

(continues on next page)

(continued from previous page)

```
|    gen/time/total_timesteps        | 65536        |
|    gen/train/approx_kl             | 0.0016218722 |
|    gen/train/clip_fraction         | 0.0488       |
|    gen/train/clip_range            | 0.1          |
|    gen/train/entropy_loss          | -0.691       |
|    gen/train/explained_variance    | 0.66         |
|    gen/train/learning_rate         | 0.0005       |
|    gen/train/loss                  | 89.1         |
|    gen/train/n_updates             | 15           |
|    gen/train/policy_gradient_loss  | -0.00034     |
|    gen/train/value_loss            | 1.37e+03     |
-----------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.782    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.564    |
|    disc/disc_entropy                | 0.666    |
|    disc/disc_loss                   | 0.62     |
|    disc/disc_proportion_expert_pred | 0.718    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.799    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.599    |
|    disc/disc_entropy                | 0.667    |
|    disc/disc_loss                   | 0.614    |
|    disc/disc_proportion_expert_pred | 0.701    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.787    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.574    |
|    disc/disc_entropy                | 0.667    |
|    disc/disc_loss                   | 0.616    |
|    disc/disc_proportion_expert_pred | 0.713    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.789    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.577    |
|    disc/disc_entropy                | 0.669    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_loss                  | 0.616    |
|    disc/disc_proportion_expert_pred | 0.711    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 4        |
|    disc/n_expert                   | 2.05e+03 |
|    disc/n_generated                | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.79     |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.58     |
|    disc/disc_entropy               | 0.668    |
|    disc/disc_loss                  | 0.612    |
|    disc/disc_proportion_expert_pred | 0.71     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 4        |
|    disc/n_expert                   | 2.05e+03 |
|    disc/n_generated                | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.812    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.623    |
|    disc/disc_entropy               | 0.669    |
|    disc/disc_loss                  | 0.604    |
|    disc/disc_proportion_expert_pred | 0.688    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 4        |
|    disc/n_expert                   | 2.05e+03 |
|    disc/n_generated                | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.806    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.612    |
|    disc/disc_entropy               | 0.669    |
|    disc/disc_loss                  | 0.6      |
|    disc/disc_proportion_expert_pred | 0.694    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 4        |
|    disc/n_expert                   | 2.05e+03 |
|    disc/n_generated                | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.798    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.597    |
|    disc/disc_entropy               | 0.671    |
|    disc/disc_loss                  | 0.605    |
|    disc/disc_proportion_expert_pred | 0.702    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 4        |
|    disc/n_expert                   | 2.05e+03 |
```

(continues on next page)

(continued from previous page)

```
|    disc/n_generated                | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.804    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.609    |
|    disc/disc_entropy               | 0.67     |
|    disc/disc_loss                  | 0.6      |
|    disc/disc_proportion_expert_pred | 0.696    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 4        |
|    disc/n_expert                   | 2.05e+03 |
|    disc/n_generated                | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.818    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.636    |
|    disc/disc_entropy               | 0.67     |
|    disc/disc_loss                  | 0.593    |
|    disc/disc_proportion_expert_pred | 0.682    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 4        |
|    disc/n_expert                   | 2.05e+03 |
|    disc/n_generated                | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.813    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.626    |
|    disc/disc_entropy               | 0.67     |
|    disc/disc_loss                  | 0.593    |
|    disc/disc_proportion_expert_pred | 0.687    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 4        |
|    disc/n_expert                   | 2.05e+03 |
|    disc/n_generated                | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.826    |
|    disc/disc_acc_expert            | 1        |
|    disc/disc_acc_gen               | 0.651    |
|    disc/disc_entropy               | 0.67     |
|    disc/disc_loss                  | 0.588    |
|    disc/disc_proportion_expert_pred | 0.674    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                | 4        |
|    disc/n_expert                   | 2.05e+03 |
|    disc/n_generated                | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                               |          |
|    disc/disc_acc                   | 0.824    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.647    |
|    disc/disc_entropy                | 0.67     |
|    disc/disc_loss                   | 0.585    |
|    disc/disc_proportion_expert_pred | 0.676    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.83     |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.66     |
|    disc/disc_entropy                | 0.669    |
|    disc/disc_loss                   | 0.581    |
|    disc/disc_proportion_expert_pred | 0.67     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.829    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.658    |
|    disc/disc_entropy                | 0.67     |
|    disc/disc_loss                   | 0.583    |
|    disc/disc_proportion_expert_pred | 0.671    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.834    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.668    |
|    disc/disc_entropy                | 0.67     |
|    disc/disc_loss                   | 0.579    |
|    disc/disc_proportion_expert_pred | 0.666    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 4        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
---------------------------------------------------
| mean/                               |           |
|    disc/disc_acc                    | 0.809     |
|    disc/disc_acc_expert             | 1         |
|    disc/disc_acc_gen                | 0.618     |
|    disc/disc_entropy                | 0.669     |
|    disc/disc_loss                   | 0.599     |
|    disc/disc_proportion_expert_pred | 0.691     |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_proportion_expert_true | 0.5       |
|    disc/global_step                 | 4         |
|    disc/n_expert                    | 2.05e+03  |
|    disc/n_generated                 | 2.05e+03  |
|    gen/rollout/ep_len_mean          | 500       |
|    gen/rollout/ep_rew_mean          | 38.2      |
|    gen/rollout/ep_rew_wrapped_mean  | -1.52e+03 |
|    gen/time/fps                     | 5.12e+03  |
|    gen/time/iterations              | 1         |
|    gen/time/time_elapsed            | 3         |
|    gen/time/total_timesteps         | 6.55e+04  |
|    gen/train/approx_kl              | 0.00297   |
|    gen/train/clip_fraction          | 0.146     |
|    gen/train/clip_range             | 0.1       |
|    gen/train/entropy_loss           | -0.687    |
|    gen/train/explained_variance     | 0.877     |
|    gen/train/learning_rate          | 0.0005    |
|    gen/train/loss                   | 4.76      |
|    gen/train/n_updates              | 20        |
|    gen/train/policy_gradient_loss   | -0.00277  |
|    gen/train/value_loss             | 266       |
---------------------------------------------------
-----------------------------------------------------
| raw/                               |              |
|    gen/rollout/ep_len_mean         | 500          |
|    gen/rollout/ep_rew_mean         | 40.5         |
|    gen/rollout/ep_rew_wrapped_mean | -1.69e+03    |
|    gen/time/fps                    | 5116         |
|    gen/time/iterations             | 1            |
|    gen/time/time_elapsed           | 3            |
|    gen/time/total_timesteps        | 81920        |
|    gen/train/approx_kl             | 0.0029702676 |
|    gen/train/clip_fraction         | 0.146        |
|    gen/train/clip_range            | 0.1          |
|    gen/train/entropy_loss          | -0.687       |
|    gen/train/explained_variance    | 0.877        |
|    gen/train/learning_rate         | 0.0005       |
|    gen/train/loss                  | 4.76         |
|    gen/train/n_updates             | 20           |
|    gen/train/policy_gradient_loss  | -0.00277     |
|    gen/train/value_loss            | 266          |
-----------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.712    |
|    disc/disc_acc_expert             | 0.998    |
|    disc/disc_acc_gen                | 0.426    |
|    disc/disc_entropy                | 0.682    |
|    disc/disc_loss                   | 0.646    |
|    disc/disc_proportion_expert_pred | 0.786    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_acc                    | 0.716    |
|    disc/disc_acc_expert             | 0.996    |
|    disc/disc_acc_gen                | 0.435    |
|    disc/disc_entropy                | 0.683    |
|    disc/disc_loss                   | 0.647    |
|    disc/disc_proportion_expert_pred | 0.781    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.728    |
|    disc/disc_acc_expert             | 0.999    |
|    disc/disc_acc_gen                | 0.457    |
|    disc/disc_entropy                | 0.683    |
|    disc/disc_loss                   | 0.644    |
|    disc/disc_proportion_expert_pred | 0.771    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.749    |
|    disc/disc_acc_expert             | 1        |
|    disc/disc_acc_gen                | 0.498    |
|    disc/disc_entropy                | 0.683    |
|    disc/disc_loss                   | 0.643    |
|    disc/disc_proportion_expert_pred | 0.751    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.767    |
|    disc/disc_acc_expert             | 0.997    |
|    disc/disc_acc_gen                | 0.537    |
|    disc/disc_entropy                | 0.684    |
|    disc/disc_loss                   | 0.637    |
|    disc/disc_proportion_expert_pred | 0.73     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.797    |
|    disc/disc_acc_expert             | 0.997    |
|    disc/disc_acc_gen                | 0.597    |
|    disc/disc_entropy                | 0.683    |
|    disc/disc_loss                   | 0.63     |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_proportion_expert_pred | 0.7      |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.807    |
|    disc/disc_acc_expert             | 0.998    |
|    disc/disc_acc_gen                | 0.617    |
|    disc/disc_entropy                | 0.683    |
|    disc/disc_loss                   | 0.63     |
|    disc/disc_proportion_expert_pred | 0.691    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.844    |
|    disc/disc_acc_expert             | 0.998    |
|    disc/disc_acc_gen                | 0.69     |
|    disc/disc_entropy                | 0.683    |
|    disc/disc_loss                   | 0.622    |
|    disc/disc_proportion_expert_pred | 0.654    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.845    |
|    disc/disc_acc_expert             | 0.999    |
|    disc/disc_acc_gen                | 0.692    |
|    disc/disc_entropy                | 0.683    |
|    disc/disc_loss                   | 0.619    |
|    disc/disc_proportion_expert_pred | 0.653    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.863    |
|    disc/disc_acc_expert             | 0.998    |
|    disc/disc_acc_gen                | 0.729    |
|    disc/disc_entropy                | 0.682    |
|    disc/disc_loss                   | 0.614    |
|    disc/disc_proportion_expert_pred | 0.635    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
```

(continues on next page)

(continued from previous page)

```
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.866    |
|    disc/disc_acc_expert             | 0.999    |
|    disc/disc_acc_gen                | 0.733    |
|    disc/disc_entropy                | 0.682    |
|    disc/disc_loss                   | 0.611    |
|    disc/disc_proportion_expert_pred | 0.633    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.866    |
|    disc/disc_acc_expert             | 0.999    |
|    disc/disc_acc_gen                | 0.733    |
|    disc/disc_entropy                | 0.682    |
|    disc/disc_loss                   | 0.609    |
|    disc/disc_proportion_expert_pred | 0.633    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.876    |
|    disc/disc_acc_expert             | 0.998    |
|    disc/disc_acc_gen                | 0.754    |
|    disc/disc_entropy                | 0.681    |
|    disc/disc_loss                   | 0.605    |
|    disc/disc_proportion_expert_pred | 0.622    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.891    |
|    disc/disc_acc_expert             | 0.999    |
|    disc/disc_acc_gen                | 0.784    |
|    disc/disc_entropy                | 0.68     |
|    disc/disc_loss                   | 0.599    |
|    disc/disc_proportion_expert_pred | 0.608    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 5        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.893    |
|    disc/disc_acc_expert             | 0.998    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_acc_gen                 | 0.788    |
|    disc/disc_entropy                 | 0.68     |
|    disc/disc_loss                    | 0.598    |
|    disc/disc_proportion_expert_pred | 0.605    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                  | 5        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                 |          |
|    disc/disc_acc                     | 0.893    |
|    disc/disc_acc_expert              | 0.997    |
|    disc/disc_acc_gen                 | 0.788    |
|    disc/disc_entropy                 | 0.68     |
|    disc/disc_loss                    | 0.597    |
|    disc/disc_proportion_expert_pred | 0.604    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                  | 5        |
|    disc/n_expert                     | 2.05e+03 |
|    disc/n_generated                  | 2.05e+03 |
--------------------------------------------------
---------------------------------------------------
| mean/                                |           |
|    disc/disc_acc                     | 0.82      |
|    disc/disc_acc_expert              | 0.998     |
|    disc/disc_acc_gen                 | 0.641     |
|    disc/disc_entropy                 | 0.682     |
|    disc/disc_loss                    | 0.622     |
|    disc/disc_proportion_expert_pred | 0.678     |
|    disc/disc_proportion_expert_true | 0.5       |
|    disc/global_step                  | 5         |
|    disc/n_expert                     | 2.05e+03  |
|    disc/n_generated                  | 2.05e+03  |
|    gen/rollout/ep_len_mean           | 500       |
|    gen/rollout/ep_rew_mean           | 40.5      |
|    gen/rollout/ep_rew_wrapped_mean   | -1.69e+03 |
|    gen/time/fps                      | 5.12e+03  |
|    gen/time/iterations               | 1         |
|    gen/time/time_elapsed             | 3         |
|    gen/time/total_timesteps          | 8.19e+04  |
|    gen/train/approx_kl               | 0.00237   |
|    gen/train/clip_fraction           | 0.136     |
|    gen/train/clip_range              | 0.1       |
|    gen/train/entropy_loss            | -0.686    |
|    gen/train/explained_variance      | 0.799     |
|    gen/train/learning_rate           | 0.0005    |
|    gen/train/loss                    | 12.1      |
|    gen/train/n_updates               | 25        |
|    gen/train/policy_gradient_loss    | -0.00326  |
|    gen/train/value_loss              | 37.5      |
---------------------------------------------------
---------------------------------------------------
| raw/                                |            |
|    gen/rollout/ep_len_mean          | 500        |
|    gen/rollout/ep_rew_mean          | 43.8       |
|    gen/rollout/ep_rew_wrapped_mean | -1.38e+03  |
```

(continues on next page)

(continued from previous page)

```
|    gen/time/fps                    | 4952       |
|    gen/time/iterations             | 1          |
|    gen/time/time_elapsed           | 3          |
|    gen/time/total_timesteps        | 98304      |
|    gen/train/approx_kl             | 0.00236941 |
|    gen/train/clip_fraction         | 0.136      |
|    gen/train/clip_range            | 0.1        |
|    gen/train/entropy_loss          | -0.686     |
|    gen/train/explained_variance    | 0.799      |
|    gen/train/learning_rate         | 0.0005     |
|    gen/train/loss                  | 12.1       |
|    gen/train/n_updates             | 25         |
|    gen/train/policy_gradient_loss  | -0.00326   |
|    gen/train/value_loss            | 37.5       |
---------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.955    |
|    disc/disc_acc_expert             | 0.94     |
|    disc/disc_acc_gen                | 0.971    |
|    disc/disc_entropy                | 0.646    |
|    disc/disc_loss                   | 0.515    |
|    disc/disc_proportion_expert_pred | 0.485    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.952    |
|    disc/disc_acc_expert             | 0.935    |
|    disc/disc_acc_gen                | 0.97     |
|    disc/disc_entropy                | 0.645    |
|    disc/disc_loss                   | 0.514    |
|    disc/disc_proportion_expert_pred | 0.483    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.942    |
|    disc/disc_acc_expert             | 0.917    |
|    disc/disc_acc_gen                | 0.967    |
|    disc/disc_entropy                | 0.646    |
|    disc/disc_loss                   | 0.515    |
|    disc/disc_proportion_expert_pred | 0.475    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.936    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_acc_expert             | 0.908    |
|    disc/disc_acc_gen                | 0.964    |
|    disc/disc_entropy                | 0.644    |
|    disc/disc_loss                   | 0.513    |
|    disc/disc_proportion_expert_pred | 0.472    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.935    |
|    disc/disc_acc_expert             | 0.906    |
|    disc/disc_acc_gen                | 0.965    |
|    disc/disc_entropy                | 0.645    |
|    disc/disc_loss                   | 0.513    |
|    disc/disc_proportion_expert_pred | 0.47     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.954    |
|    disc/disc_acc_expert             | 0.936    |
|    disc/disc_acc_gen                | 0.971    |
|    disc/disc_entropy                | 0.643    |
|    disc/disc_loss                   | 0.509    |
|    disc/disc_proportion_expert_pred | 0.482    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.948    |
|    disc/disc_acc_expert             | 0.93     |
|    disc/disc_acc_gen                | 0.966    |
|    disc/disc_entropy                | 0.642    |
|    disc/disc_loss                   | 0.508    |
|    disc/disc_proportion_expert_pred | 0.482    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.956    |
|    disc/disc_acc_expert             | 0.944    |
|    disc/disc_acc_gen                | 0.968    |
|    disc/disc_entropy                | 0.642    |
|    disc/disc_loss                   | 0.506    |
|    disc/disc_proportion_expert_pred | 0.488    |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.958    |
|    disc/disc_acc_expert             | 0.957    |
|    disc/disc_acc_gen                | 0.958    |
|    disc/disc_entropy                | 0.639    |
|    disc/disc_loss                   | 0.503    |
|    disc/disc_proportion_expert_pred | 0.499    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.963    |
|    disc/disc_acc_expert             | 0.959    |
|    disc/disc_acc_gen                | 0.967    |
|    disc/disc_entropy                | 0.639    |
|    disc/disc_loss                   | 0.501    |
|    disc/disc_proportion_expert_pred | 0.496    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.963    |
|    disc/disc_acc_expert             | 0.965    |
|    disc/disc_acc_gen                | 0.961    |
|    disc/disc_entropy                | 0.639    |
|    disc/disc_loss                   | 0.5      |
|    disc/disc_proportion_expert_pred | 0.502    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.97     |
|    disc/disc_acc_expert             | 0.973    |
|    disc/disc_acc_gen                | 0.968    |
|    disc/disc_entropy                | 0.636    |
|    disc/disc_loss                   | 0.495    |
|    disc/disc_proportion_expert_pred | 0.502    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.973    |
|    disc/disc_acc_expert             | 0.979    |
|    disc/disc_acc_gen                | 0.966    |
|    disc/disc_entropy                | 0.637    |
|    disc/disc_loss                   | 0.497    |
|    disc/disc_proportion_expert_pred | 0.507    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.97     |
|    disc/disc_acc_expert             | 0.971    |
|    disc/disc_acc_gen                | 0.969    |
|    disc/disc_entropy                | 0.634    |
|    disc/disc_loss                   | 0.493    |
|    disc/disc_proportion_expert_pred | 0.501    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.978    |
|    disc/disc_acc_expert             | 0.988    |
|    disc/disc_acc_gen                | 0.968    |
|    disc/disc_entropy                | 0.635    |
|    disc/disc_loss                   | 0.494    |
|    disc/disc_proportion_expert_pred | 0.51     |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
--------------------------------------------------
| raw/                                |          |
|    disc/disc_acc                    | 0.976    |
|    disc/disc_acc_expert             | 0.984    |
|    disc/disc_acc_gen                | 0.967    |
|    disc/disc_entropy                | 0.634    |
|    disc/disc_loss                   | 0.49     |
|    disc/disc_proportion_expert_pred | 0.509    |
|    disc/disc_proportion_expert_true | 0.5      |
|    disc/global_step                 | 6        |
|    disc/n_expert                    | 2.05e+03 |
|    disc/n_generated                 | 2.05e+03 |
--------------------------------------------------
---------------------------------------------------
| mean/                               |           |
|    disc/disc_acc                    | 0.958     |
|    disc/disc_acc_expert             | 0.95      |
|    disc/disc_acc_gen                | 0.967     |
```

(continues on next page)

(continued from previous page)

```
|    disc/disc_entropy                | 0.64      |
|    disc/disc_loss                   | 0.504     |
|    disc/disc_proportion_expert_pred | 0.491     |
|    disc/disc_proportion_expert_true | 0.5       |
|    disc/global_step                 | 6         |
|    disc/n_expert                    | 2.05e+03  |
|    disc/n_generated                 | 2.05e+03  |
|    gen/rollout/ep_len_mean          | 500       |
|    gen/rollout/ep_rew_mean          | 43.8      |
|    gen/rollout/ep_rew_wrapped_mean  | -1.38e+03 |
|    gen/time/fps                     | 4.95e+03  |
|    gen/time/iterations              | 1         |
|    gen/time/time_elapsed            | 3         |
|    gen/time/total_timesteps         | 9.83e+04  |
|    gen/train/approx_kl              | 0.002     |
|    gen/train/clip_fraction          | 0.0806    |
|    gen/train/clip_range             | 0.1       |
|    gen/train/entropy_loss           | -0.684    |
|    gen/train/explained_variance     | 0.47      |
|    gen/train/learning_rate          | 0.0005    |
|    gen/train/loss                   | 2.66      |
|    gen/train/n_updates              | 30        |
|    gen/train/policy_gradient_loss   | -0.00117  |
|    gen/train/value_loss             | 94.9      |
---------------------------------------------------
```

We can see that an untrained policy performs poorly, while AIRL brings an improvement. To make it match the expert performance (500), set the flag `FAST` to `False` in the first cell.

```
print(
    "Rewards before training:",
    np.mean(learner_rewards_before_training),
    "+/-",
    np.std(learner_rewards_before_training),
)
print(
    "Rewards after training:",
    np.mean(learner_rewards_after_training),
    "+/-",
    np.std(learner_rewards_after_training),
)
```

```
Rewards before training: 102.6 +/- 24.11514047232568
Rewards after training: 43.02 +/- 3.4379645140693347
```

download this notebook here

## 2.23 Learning a Reward Function using Preference Comparisons

The preference comparisons algorithm learns a reward function by comparing trajectory segments to each other.

To set up the preference comparisons algorithm, we first need to set up a lot of its internals beforehand:

```
import random
from imitation.algorithms import preference_comparisons
from imitation.rewards.reward_nets import BasicRewardNet
from imitation.util.networks import RunningNorm
from imitation.util.util import make_vec_env
from imitation.policies.base import FeedForward32Policy, NormalizeFeaturesExtractor
import gymnasium as gym
from stable_baselines3 import PPO
import numpy as np

rng = np.random.default_rng(0)

venv = make_vec_env("Pendulum-v1", rng=rng)

reward_net = BasicRewardNet(
    venv.observation_space, venv.action_space, normalize_input_layer=RunningNorm
)

fragmenter = preference_comparisons.RandomFragmenter(
    warning_threshold=0,
    rng=rng,
)
gatherer = preference_comparisons.SyntheticGatherer(rng=rng)
preference_model = preference_comparisons.PreferenceModel(reward_net)
reward_trainer = preference_comparisons.BasicRewardTrainer(
    preference_model=preference_model,
    loss=preference_comparisons.CrossEntropyRewardLoss(),
    epochs=3,
    rng=rng,
)


# Several hyperparameters (reward_epochs, ppo_clip_range, ppo_ent_coef,
# ppo_gae_lambda, ppo_n_epochs, discount_factor, use_sde, sde_sample_freq,
# ppo_lr, exploration_frac, num_iterations, initial_comparison_frac,
# initial_epoch_multiplier, query_schedule) used in this example have been
# approximately fine-tuned to reach a reasonable level of performance.
agent = PPO(
    policy=FeedForward32Policy,
    policy_kwargs=dict(
        features_extractor_class=NormalizeFeaturesExtractor,
        features_extractor_kwargs=dict(normalize_class=RunningNorm),
    ),
    env=venv,
    seed=0,
    n_steps=2048 // venv.num_envs,
    batch_size=64,
    ent_coef=0.01,
    learning_rate=2e-3,
    clip_range=0.1,
    gae_lambda=0.95,
```

(continues on next page)

(continued from previous page)

```
    gamma=0.97,
    n_epochs=10,
)

trajectory_generator = preference_comparisons.AgentTrainer(
    algorithm=agent,
    reward_fn=reward_net,
    venv=venv,
    exploration_frac=0.05,
    rng=rng,
)

pref_comparisons = preference_comparisons.PreferenceComparisons(
    trajectory_generator,
    reward_net,
    num_iterations=5,  # Set to 60 for better performance
    fragmenter=fragmenter,
    preference_gatherer=gatherer,
    reward_trainer=reward_trainer,
    fragment_length=100,
    transition_oversampling=1,
    initial_comparison_frac=0.1,
    allow_variable_horizon=False,
    initial_epoch_multiplier=4,
    query_schedule="hyperbolic",
)
```

Then we can start training the reward model. Note that we need to specify the total timesteps that the agent should be trained and how many fragment comparisons should be made.

```
pref_comparisons.train(
    total_timesteps=5_000,
    total_comparisons=200,
)
```

```
Query schedule: [20, 51, 41, 34, 29, 25]
Collecting 40 fragments (4000 transitions)
Requested 3800 transitions but only 0 in buffer. Sampling 3800 additional transitions.
Sampling 200 exploratory transitions.
Creating fragment pairs
Gathering preferences
Dataset now contains 20 comparisons
Training agent for 1000 timesteps
---------------------------------------------------
| raw/                                 |          |
|    agent/rollout/ep_len_mean         | 200      |
|    agent/rollout/ep_rew_mean         | -1.2e+03 |
|    agent/rollout/ep_rew_wrapped_mean | -64.3    |
|    agent/time/fps                    | 5857     |
|    agent/time/iterations             | 1        |
|    agent/time/time_elapsed           | 0        |
|    agent/time/total_timesteps        | 2048     |
---------------------------------------------------
------------------------------------------------------
| mean/                                    |          |
|    agent/rollout/ep_len_mean             | 200      |
```

(continues on next page)

(continued from previous page)

```
|    agent/rollout/ep_rew_mean            | -1.2e+03 |
|    agent/rollout/ep_rew_wrapped_mean    | -64.3    |
|    agent/time/fps                       | 5.86e+03 |
|    agent/time/iterations                | 1        |
|    agent/time/time_elapsed              | 0        |
|    agent/time/total_timesteps           | 2.05e+03 |
|    agent/train/approx_kl                | 0.00256  |
|    agent/train/clip_fraction            | 0.0969   |
|    agent/train/clip_range               | 0.1      |
|    agent/train/entropy_loss             | -1.44    |
|    agent/train/explained_variance       | -0.267   |
|    agent/train/learning_rate            | 0.002    |
|    agent/train/loss                     | 0.144    |
|    agent/train/n_updates                | 10       |
|    agent/train/policy_gradient_loss     | -0.00352 |
|    agent/train/std                      | 1.02     |
|    agent/train/value_loss               | 0.572    |
|    preferences/entropy                  | 0.0307   |
|    reward/epoch-0/train/accuracy        | 0.6      |
|    reward/epoch-0/train/gt_reward_loss  | 0.0639   |
|    reward/epoch-0/train/loss            | 0.71     |
|    reward/epoch-1/train/accuracy        | 0.55     |
|    reward/epoch-1/train/gt_reward_loss  | 0.0639   |
|    reward/epoch-1/train/loss            | 0.648    |
|    reward/epoch-10/train/accuracy       | 0.9      |
|    reward/epoch-10/train/gt_reward_loss | 0.0639   |
|    reward/epoch-10/train/loss           | 0.198    |
|    reward/epoch-11/train/accuracy       | 0.9      |
|    reward/epoch-11/train/gt_reward_loss | 0.0639   |
|    reward/epoch-11/train/loss           | 0.185    |
|    reward/epoch-2/train/accuracy        | 0.6      |
|    reward/epoch-2/train/gt_reward_loss  | 0.0639   |
|    reward/epoch-2/train/loss            | 0.554    |
|    reward/epoch-3/train/accuracy        | 0.65     |
|    reward/epoch-3/train/gt_reward_loss  | 0.0639   |
|    reward/epoch-3/train/loss            | 0.454    |
|    reward/epoch-4/train/accuracy        | 0.85     |
|    reward/epoch-4/train/gt_reward_loss  | 0.0639   |
|    reward/epoch-4/train/loss            | 0.383    |
|    reward/epoch-5/train/accuracy        | 0.85     |
|    reward/epoch-5/train/gt_reward_loss  | 0.0639   |
|    reward/epoch-5/train/loss            | 0.326    |
|    reward/epoch-6/train/accuracy        | 0.9      |
|    reward/epoch-6/train/gt_reward_loss  | 0.0639   |
|    reward/epoch-6/train/loss            | 0.285    |
|    reward/epoch-7/train/accuracy        | 0.9      |
|    reward/epoch-7/train/gt_reward_loss  | 0.0639   |
|    reward/epoch-7/train/loss            | 0.255    |
|    reward/epoch-8/train/accuracy        | 0.9      |
|    reward/epoch-8/train/gt_reward_loss  | 0.0639   |
|    reward/epoch-8/train/loss            | 0.231    |
|    reward/epoch-9/train/accuracy        | 0.9      |
|    reward/epoch-9/train/gt_reward_loss  | 0.0639   |
|    reward/epoch-9/train/loss            | 0.213    |
| reward/                                 |          |
|    final/train/accuracy                 | 0.9      |
|    final/train/gt_reward_loss           | 0.0639   |
```

(continues on next page)

(continued from previous page)

```
|    final/train/loss                     | 0.185    |
------------------------------------------------------
Collecting 102 fragments (10200 transitions)
Requested 9690 transitions but only 1600 in buffer. Sampling 8090 additional␣
↪transitions.
Sampling 510 exploratory transitions.
Creating fragment pairs
Gathering preferences
Dataset now contains 71 comparisons
Training agent for 1000 timesteps
-------------------------------------------------------
| raw/                                 |              |
|    agent/rollout/ep_len_mean         | 200          |
|    agent/rollout/ep_rew_mean         | -1.14e+03    |
|    agent/rollout/ep_rew_wrapped_mean | -42.6        |
|    agent/time/fps                    | 5895         |
|    agent/time/iterations             | 1            |
|    agent/time/time_elapsed           | 0            |
|    agent/time/total_timesteps        | 4096         |
|    agent/train/approx_kl             | 0.0025575275 |
|    agent/train/clip_fraction         | 0.0969       |
|    agent/train/clip_range            | 0.1          |
|    agent/train/entropy_loss          | -1.44        |
|    agent/train/explained_variance    | -0.267       |
|    agent/train/learning_rate         | 0.002        |
|    agent/train/loss                  | 0.144        |
|    agent/train/n_updates             | 10           |
|    agent/train/policy_gradient_loss  | -0.00352     |
|    agent/train/std                   | 1.02         |
|    agent/train/value_loss            | 0.572        |
-------------------------------------------------------
------------------------------------------------------
| mean/                                  |           |
|    agent/rollout/ep_len_mean           | 200       |
|    agent/rollout/ep_rew_mean           | -1.14e+03 |
|    agent/rollout/ep_rew_wrapped_mean   | -42.6     |
|    agent/time/fps                      | 5.9e+03   |
|    agent/time/iterations               | 1         |
|    agent/time/time_elapsed             | 0         |
|    agent/time/total_timesteps          | 4.1e+03   |
|    agent/train/approx_kl               | 0.0026    |
|    agent/train/clip_fraction           | 0.113     |
|    agent/train/clip_range              | 0.1       |
|    agent/train/entropy_loss            | -1.45     |
|    agent/train/explained_variance      | 0.729     |
|    agent/train/learning_rate           | 0.002     |
|    agent/train/loss                    | 0.0361    |
|    agent/train/n_updates               | 20        |
|    agent/train/policy_gradient_loss    | -0.00576  |
|    agent/train/std                     | 1.03      |
|    agent/train/value_loss              | 0.214     |
|    preferences/entropy                 | 0.00723   |
|    reward/epoch-0/train/accuracy       | 0.938     |
|    reward/epoch-0/train/gt_reward_loss | 0.0146    |
|    reward/epoch-0/train/loss           | 0.13      |
|    reward/epoch-1/train/accuracy       | 0.938     |
|    reward/epoch-1/train/gt_reward_loss | 0.0146    |
```

(continues on next page)

(continued from previous page)

```
|    reward/epoch-1/train/loss           | 0.111     |
|    reward/epoch-2/train/accuracy       | 0.911     |
|    reward/epoch-2/train/gt_reward_loss | 0.0146    |
|    reward/epoch-2/train/loss           | 0.125     |
| reward/                                |           |
|    final/train/accuracy                | 0.911     |
|    final/train/gt_reward_loss          | 0.0146    |
|    final/train/loss                    | 0.125     |
------------------------------------------------------
Collecting 82 fragments (8200 transitions)
Requested 7790 transitions but only 1600 in buffer. Sampling 6190 additional␣
↪transitions.
Sampling 410 exploratory transitions.
Creating fragment pairs
Gathering preferences
Dataset now contains 112 comparisons
Training agent for 1000 timesteps
------------------------------------------------------
| raw/                                 |             |
|    agent/rollout/ep_len_mean         | 200         |
|    agent/rollout/ep_rew_mean         | -1.16e+03   |
|    agent/rollout/ep_rew_wrapped_mean | -28         |
|    agent/time/fps                    | 5863        |
|    agent/time/iterations             | 1           |
|    agent/time/time_elapsed           | 0           |
|    agent/time/total_timesteps        | 6144        |
|    agent/train/approx_kl             | 0.002599684 |
|    agent/train/clip_fraction         | 0.113       |
|    agent/train/clip_range            | 0.1         |
|    agent/train/entropy_loss          | -1.45       |
|    agent/train/explained_variance    | 0.729       |
|    agent/train/learning_rate         | 0.002       |
|    agent/train/loss                  | 0.0361      |
|    agent/train/n_updates             | 20          |
|    agent/train/policy_gradient_loss  | -0.00576    |
|    agent/train/std                   | 1.03        |
|    agent/train/value_loss            | 0.214       |
------------------------------------------------------
------------------------------------------------------
| mean/                                  |           |
|    agent/rollout/ep_len_mean           | 200       |
|    agent/rollout/ep_rew_mean           | -1.16e+03 |
|    agent/rollout/ep_rew_wrapped_mean   | -28       |
|    agent/time/fps                      | 5.86e+03  |
|    agent/time/iterations               | 1         |
|    agent/time/time_elapsed             | 0         |
|    agent/time/total_timesteps          | 6.14e+03  |
|    agent/train/approx_kl               | 0.00228   |
|    agent/train/clip_fraction           | 0.108     |
|    agent/train/clip_range              | 0.1       |
|    agent/train/entropy_loss            | -1.45     |
|    agent/train/explained_variance      | 0.854     |
|    agent/train/learning_rate           | 0.002     |
|    agent/train/loss                    | 0.0487    |
|    agent/train/n_updates               | 30        |
|    agent/train/policy_gradient_loss    | -0.00489  |
|    agent/train/std                     | 1.03      |
```

(continues on next page)

(continued from previous page)

```
|    agent/train/value_loss             | 0.144     |
|    preferences/entropy                | 0.00114   |
|    reward/epoch-0/train/accuracy      | 0.922     |
|    reward/epoch-0/train/gt_reward_loss | 0.0111    |
|    reward/epoch-0/train/loss          | 0.176     |
|    reward/epoch-1/train/accuracy      | 0.93      |
|    reward/epoch-1/train/gt_reward_loss | 0.011     |
|    reward/epoch-1/train/loss          | 0.145     |
|    reward/epoch-2/train/accuracy      | 0.93      |
|    reward/epoch-2/train/gt_reward_loss | 0.011     |
|    reward/epoch-2/train/loss          | 0.124     |
| reward/                               |           |
|    final/train/accuracy               | 0.93      |
|    final/train/gt_reward_loss         | 0.011     |
|    final/train/loss                   | 0.124     |
------------------------------------------------------
Collecting 68 fragments (6800 transitions)
Requested 6460 transitions but only 1600 in buffer. Sampling 4860 additional␣
↪transitions.
Sampling 340 exploratory transitions.
Creating fragment pairs
Gathering preferences
Dataset now contains 146 comparisons
Training agent for 1000 timesteps
-------------------------------------------------------
| raw/                                 |              |
|    agent/rollout/ep_len_mean         | 200          |
|    agent/rollout/ep_rew_mean         | -1.19e+03    |
|    agent/rollout/ep_rew_wrapped_mean | -22.8        |
|    agent/time/fps                    | 5871         |
|    agent/time/iterations             | 1            |
|    agent/time/time_elapsed           | 0            |
|    agent/time/total_timesteps        | 8192         |
|    agent/train/approx_kl             | 0.0022802677 |
|    agent/train/clip_fraction         | 0.108        |
|    agent/train/clip_range            | 0.1          |
|    agent/train/entropy_loss          | -1.45        |
|    agent/train/explained_variance    | 0.854        |
|    agent/train/learning_rate         | 0.002        |
|    agent/train/loss                  | 0.0487       |
|    agent/train/n_updates             | 30           |
|    agent/train/policy_gradient_loss  | -0.00489     |
|    agent/train/std                   | 1.03         |
|    agent/train/value_loss            | 0.144        |
-------------------------------------------------------
------------------------------------------------------
| mean/                                 |           |
|    agent/rollout/ep_len_mean          | 200       |
|    agent/rollout/ep_rew_mean          | -1.19e+03 |
|    agent/rollout/ep_rew_wrapped_mean  | -22.8     |
|    agent/time/fps                     | 5.87e+03  |
|    agent/time/iterations              | 1         |
|    agent/time/time_elapsed            | 0         |
|    agent/time/total_timesteps         | 8.19e+03  |
|    agent/train/approx_kl              | 0.00239   |
|    agent/train/clip_fraction          | 0.112     |
|    agent/train/clip_range             | 0.1       |
```

(continues on next page)

(continued from previous page)

```
|    agent/train/entropy_loss            | -1.46      |
|    agent/train/explained_variance      | 0.913      |
|    agent/train/learning_rate           | 0.002      |
|    agent/train/loss                    | 0.0118     |
|    agent/train/n_updates               | 40         |
|    agent/train/policy_gradient_loss    | -0.00509   |
|    agent/train/std                     | 1.05       |
|    agent/train/value_loss              | 0.156      |
|    preferences/entropy                 | 0.019      |
|    reward/epoch-0/train/accuracy       | 0.951      |
|    reward/epoch-0/train/gt_reward_loss | 0.0177     |
|    reward/epoch-0/train/loss           | 0.106      |
|    reward/epoch-1/train/accuracy       | 0.969      |
|    reward/epoch-1/train/gt_reward_loss | 0.0116     |
|    reward/epoch-1/train/loss           | 0.0849     |
|    reward/epoch-2/train/accuracy       | 0.97       |
|    reward/epoch-2/train/gt_reward_loss | 0.0136     |
|    reward/epoch-2/train/loss           | 0.0865     |
| reward/                                |            |
|    final/train/accuracy                | 0.97       |
|    final/train/gt_reward_loss          | 0.0136     |
|    final/train/loss                    | 0.0865     |
------------------------------------------------------
Collecting 58 fragments (5800 transitions)
Requested 5510 transitions but only 1600 in buffer. Sampling 3910 additional␣
 ↪transitions.
Sampling 290 exploratory transitions.
Creating fragment pairs
Gathering preferences
Dataset now contains 175 comparisons
Training agent for 1000 timesteps
-------------------------------------------------------
| raw/                                 |              |
|    agent/rollout/ep_len_mean         | 200          |
|    agent/rollout/ep_rew_mean         | -1.21e+03    |
|    agent/rollout/ep_rew_wrapped_mean | -20.3        |
|    agent/time/fps                    | 5859         |
|    agent/time/iterations             | 1            |
|    agent/time/time_elapsed           | 0            |
|    agent/time/total_timesteps        | 10240        |
|    agent/train/approx_kl             | 0.0023874408 |
|    agent/train/clip_fraction         | 0.112        |
|    agent/train/clip_range            | 0.1          |
|    agent/train/entropy_loss          | -1.46        |
|    agent/train/explained_variance    | 0.913        |
|    agent/train/learning_rate         | 0.002        |
|    agent/train/loss                  | 0.0118       |
|    agent/train/n_updates             | 40           |
|    agent/train/policy_gradient_loss  | -0.00509     |
|    agent/train/std                   | 1.05         |
|    agent/train/value_loss            | 0.156        |
-------------------------------------------------------
------------------------------------------------------
| mean/                                  |            |
|    agent/rollout/ep_len_mean           | 200        |
|    agent/rollout/ep_rew_mean           | -1.21e+03 |
|    agent/rollout/ep_rew_wrapped_mean   | -20.3      |
```

(continues on next page)

(continued from previous page)

```
|    agent/time/fps                      | 5.86e+03  |
|    agent/time/iterations               | 1         |
|    agent/time/time_elapsed             | 0         |
|    agent/time/total_timesteps          | 1.02e+04  |
|    agent/train/approx_kl               | 0.00431   |
|    agent/train/clip_fraction           | 0.197     |
|    agent/train/clip_range              | 0.1       |
|    agent/train/entropy_loss            | -1.49     |
|    agent/train/explained_variance      | 0.95      |
|    agent/train/learning_rate           | 0.002     |
|    agent/train/loss                    | 0.0066    |
|    agent/train/n_updates               | 50        |
|    agent/train/policy_gradient_loss    | -0.0116   |
|    agent/train/std                     | 1.08      |
|    agent/train/value_loss              | 0.168     |
|    preferences/entropy                 | 4.1e-07   |
|    reward/epoch-0/train/accuracy       | 0.979     |
|    reward/epoch-0/train/gt_reward_loss | 0.00959   |
|    reward/epoch-0/train/loss           | 0.069     |
|    reward/epoch-1/train/accuracy       | 0.984     |
|    reward/epoch-1/train/gt_reward_loss | 0.00959   |
|    reward/epoch-1/train/loss           | 0.0644    |
|    reward/epoch-2/train/accuracy       | 0.984     |
|    reward/epoch-2/train/gt_reward_loss | 0.0121    |
|    reward/epoch-2/train/loss           | 0.0645    |
| reward/                                |           |
|    final/train/accuracy                | 0.984     |
|    final/train/gt_reward_loss          | 0.0121    |
|    final/train/loss                    | 0.0645    |
------------------------------------------------------
Collecting 50 fragments (5000 transitions)
Requested 4750 transitions but only 1600 in buffer. Sampling 3150 additional␣
↪transitions.
Sampling 250 exploratory transitions.
Creating fragment pairs
Gathering preferences
Dataset now contains 200 comparisons
Training agent for 1000 timesteps
------------------------------------------------------
| raw/                                 |             |
|    agent/rollout/ep_len_mean         | 200         |
|    agent/rollout/ep_rew_mean         | -1.2e+03    |
|    agent/rollout/ep_rew_wrapped_mean | -20.2       |
|    agent/time/fps                    | 5803        |
|    agent/time/iterations             | 1           |
|    agent/time/time_elapsed           | 0           |
|    agent/time/total_timesteps        | 12288       |
|    agent/train/approx_kl             | 0.004307774 |
|    agent/train/clip_fraction         | 0.197       |
|    agent/train/clip_range            | 0.1         |
|    agent/train/entropy_loss          | -1.49       |
|    agent/train/explained_variance    | 0.95        |
|    agent/train/learning_rate         | 0.002       |
|    agent/train/loss                  | 0.0066      |
|    agent/train/n_updates             | 50          |
|    agent/train/policy_gradient_loss  | -0.0116     |
|    agent/train/std                   | 1.08        |
```

(continues on next page)

(continued from previous page)

```
|    agent/train/value_loss            | 0.168       |
------------------------------------------------------
-----------------------------------------------------
| mean/                                 |          |
|    agent/rollout/ep_len_mean          | 200      |
|    agent/rollout/ep_rew_mean          | -1.2e+03 |
|    agent/rollout/ep_rew_wrapped_mean  | -20.2    |
|    agent/time/fps                     | 5.8e+03  |
|    agent/time/iterations              | 1        |
|    agent/time/time_elapsed            | 0        |
|    agent/time/total_timesteps         | 1.23e+04 |
|    agent/train/approx_kl              | 0.00304  |
|    agent/train/clip_fraction          | 0.137    |
|    agent/train/clip_range             | 0.1      |
|    agent/train/entropy_loss           | -1.5     |
|    agent/train/explained_variance     | 0.962    |
|    agent/train/learning_rate          | 0.002    |
|    agent/train/loss                   | 0.237    |
|    agent/train/n_updates              | 60       |
|    agent/train/policy_gradient_loss   | -0.00633 |
|    agent/train/std                    | 1.09     |
|    agent/train/value_loss             | 0.182    |
|    preferences/entropy                | 0.00186  |
|    reward/epoch-0/train/accuracy      | 0.955    |
|    reward/epoch-0/train/gt_reward_loss | 0.00826  |
|    reward/epoch-0/train/loss          | 0.152    |
|    reward/epoch-1/train/accuracy      | 0.973    |
|    reward/epoch-1/train/gt_reward_loss | 0.00826  |
|    reward/epoch-1/train/loss          | 0.111    |
|    reward/epoch-2/train/accuracy      | 0.973    |
|    reward/epoch-2/train/gt_reward_loss | 0.00996  |
|    reward/epoch-2/train/loss          | 0.103    |
| reward/                               |          |
|    final/train/accuracy               | 0.973    |
|    final/train/gt_reward_loss         | 0.00996  |
|    final/train/loss                   | 0.103    |
-----------------------------------------------------
```

```
{'reward_loss': 0.10251256078481674, 'reward_accuracy': 0.9732142857142858}
```

After we trained the reward network using the preference comparisons algorithm, we can wrap our environment with that learned reward.

```python
from imitation.rewards.reward_wrapper import RewardVecEnvWrapper

learned_reward_venv = RewardVecEnvWrapper(venv, reward_net.predict_processed)
```

Next, we train an agent that sees only the shaped, learned reward.

```python
learner = PPO(
    seed=0,
    policy=FeedForward32Policy,
    policy_kwargs=dict(
        features_extractor_class=NormalizeFeaturesExtractor,
        features_extractor_kwargs=dict(normalize_class=RunningNorm),
    ),
```

(continues on next page)

(continued from previous page)

```
    env=learned_reward_venv,
    batch_size=64,
    ent_coef=0.01,
    n_epochs=10,
    n_steps=2048 // learned_reward_venv.num_envs,
    clip_range=0.1,
    gae_lambda=0.95,
    gamma=0.97,
    learning_rate=2e-3,
)
learner.learn(1_000)  # Note: set to 100_000 to train a proficient expert
```

```
<stable_baselines3.ppo.ppo.PPO at 0x7f84f8099700>
```

Then we can evaluate it using the original reward.

```
from stable_baselines3.common.evaluation import evaluate_policy

n_eval_episodes = 10
reward_mean, reward_std = evaluate_policy(learner.policy, venv, n_eval_episodes)
reward_stderr = reward_std / np.sqrt(n_eval_episodes)
print(f"Reward: {reward_mean:.0f} +/- {reward_stderr:.0f}")
```

```
Reward: -1332 +/- 119
```

download this notebook here

## 2.24 Learning a Reward Function using Preference Comparisons on Atari

In this case, we will use a convolutional neural network for our policy and reward model. We will also shape the learned reward model with the policy's learned value function, since these shaped rewards will be more informative for training - incentivizing agents to move to high-value states. In the interests of execution time, we will only do a little bit of training - much less than in the previous preference comparison notebook. To run this notebook, be sure to install the `atari` extras, for example by running `pip install imitation[atari]`.

First, we will set up the environment, reward network, et cetera.

```
import torch as th
import gymnasium as gym
from gymnasium.wrappers import TimeLimit
import numpy as np

from seals.util import AutoResetWrapper

from stable_baselines3 import PPO
from stable_baselines3.common.atari_wrappers import AtariWrapper
from stable_baselines3.common.env_util import make_vec_env
from stable_baselines3.common.vec_env import VecFrameStack
from stable_baselines3.ppo import CnnPolicy

from imitation.algorithms import preference_comparisons
from imitation.data.wrappers import RolloutInfoWrapper
```

(continues on next page)

(continued from previous page)

```
from imitation.policies.base import NormalizeFeaturesExtractor
from imitation.rewards.reward_nets import CnnRewardNet


device = th.device("cuda" if th.cuda.is_available() else "cpu")

rng = np.random.default_rng()


# Here we ensure that our environment has constant-length episodes by resetting
# it when done, and running until 100 timesteps have elapsed.
# For real training, you will want a much longer time limit.
def constant_length_asteroids(num_steps):
    atari_env = gym.make("AsteroidsNoFrameskip-v4")
    preprocessed_env = AtariWrapper(atari_env)
    endless_env = AutoResetWrapper(preprocessed_env)
    limited_env = TimeLimit(endless_env, max_episode_steps=num_steps)
    return RolloutInfoWrapper(limited_env)


# For real training, you will want a vectorized environment with 8 environments in␣
↪parallel.
# This can be done by passing in n_envs=8 as an argument to make_vec_env.
# The seed needs to be set to 1 for reproducibility and also to avoid win32
# np.random.randint high bound error.
venv = make_vec_env(constant_length_asteroids, env_kwargs={"num_steps": 100}, seed=1)
venv = VecFrameStack(venv, n_stack=4)

reward_net = CnnRewardNet(
    venv.observation_space,
    venv.action_space,
).to(device)

fragmenter = preference_comparisons.RandomFragmenter(warning_threshold=0, rng=rng)
gatherer = preference_comparisons.SyntheticGatherer(rng=rng)
preference_model = preference_comparisons.PreferenceModel(reward_net)
reward_trainer = preference_comparisons.BasicRewardTrainer(
    preference_model=preference_model,
    loss=preference_comparisons.CrossEntropyRewardLoss(),
    epochs=3,
    rng=rng,
)

agent = PPO(
    policy=CnnPolicy,
    env=venv,
    seed=0,
    n_steps=16,  # To train on atari well, set this to 128
    batch_size=16,  # To train on atari well, set this to 256
    ent_coef=0.01,
    learning_rate=0.00025,
    n_epochs=4,
)

trajectory_generator = preference_comparisons.AgentTrainer(
    algorithm=agent,
    reward_fn=reward_net,
```

(continues on next page)

(continued from previous page)

```
    venv=venv,
    exploration_frac=0.0,
    rng=rng,
)

pref_comparisons = preference_comparisons.PreferenceComparisons(
    trajectory_generator,
    reward_net,
    num_iterations=2,
    fragmenter=fragmenter,
    preference_gatherer=gatherer,
    reward_trainer=reward_trainer,
    fragment_length=10,
    transition_oversampling=1,
    initial_comparison_frac=0.1,
    allow_variable_horizon=False,
    initial_epoch_multiplier=1,
)
```

We are now ready to train the reward model.

```
pref_comparisons.train(
    total_timesteps=16,
    total_comparisons=15,
)
```

```
Query schedule: [1, 9, 5]
Collecting 2 fragments (20 transitions)
Requested 20 transitions but only 0 in buffer. Sampling 20 additional transitions.
Creating fragment pairs
Gathering preferences
Dataset now contains 1 comparisons
Training agent for 8 timesteps
---------------------------------------------------
| raw/                                 |          |
|    agent/rollout/ep_rew_wrapped_mean | 2.82     |
|    agent/time/fps                    | 212      |
|    agent/time/iterations             | 1        |
|    agent/time/time_elapsed           | 0        |
|    agent/time/total_timesteps        | 16       |
---------------------------------------------------
-----------------------------------------------------
| mean/                                  |          |
|    agent/rollout/ep_rew_wrapped_mean   | 2.82     |
|    agent/time/fps                      | 212      |
|    agent/time/iterations               | 1        |
|    agent/time/time_elapsed             | 0        |
|    agent/time/total_timesteps          | 16       |
|    agent/train/approx_kl               | 0.000105 |
|    agent/train/clip_fraction           | 0        |
|    agent/train/clip_range              | 0.2      |
|    agent/train/entropy_loss            | -2.64    |
|    agent/train/explained_variance      | 0.0699   |
|    agent/train/learning_rate           | 0.00025  |
|    agent/train/loss                    | -0.024   |
|    agent/train/n_updates               | 4        |
```

(continues on next page)

(continued from previous page)

```
|    agent/train/policy_gradient_loss    | -0.0054  |
|    agent/train/value_loss              | 0.0873   |
|    preferences/entropy                 | 0.693    |
|    reward/epoch-0/train/accuracy       | 1        |
|    reward/epoch-0/train/gt_reward_loss | 0.693    |
|    reward/epoch-0/train/loss           | 0.404    |
|    reward/epoch-1/train/accuracy       | 1        |
|    reward/epoch-1/train/gt_reward_loss | 0.693    |
|    reward/epoch-1/train/loss           | 0.367    |
|    reward/epoch-2/train/accuracy       | 1        |
|    reward/epoch-2/train/gt_reward_loss | 0.693    |
|    reward/epoch-2/train/loss           | 0.332    |
| reward/                                |          |
|    final/train/accuracy                | 1        |
|    final/train/gt_reward_loss          | 0.693    |
|    final/train/loss                    | 0.332    |
-----------------------------------------------------
Collecting 18 fragments (180 transitions)
Requested 180 transitions but only 0 in buffer. Sampling 180 additional transitions.
Creating fragment pairs
Gathering preferences
Dataset now contains 10 comparisons
Training agent for 8 timesteps
--------------------------------------------------------
| raw/                                 |               |
|    agent/rollout/ep_rew_wrapped_mean | 2.67          |
|    agent/time/fps                    | 173           |
|    agent/time/iterations             | 1             |
|    agent/time/time_elapsed           | 0             |
|    agent/time/total_timesteps        | 32            |
|    agent/train/approx_kl             | 0.00010485947 |
|    agent/train/clip_fraction         | 0             |
|    agent/train/clip_range            | 0.2           |
|    agent/train/entropy_loss          | -2.64         |
|    agent/train/explained_variance    | 0.0699        |
|    agent/train/learning_rate         | 0.00025       |
|    agent/train/loss                  | -0.024        |
|    agent/train/n_updates             | 4             |
|    agent/train/policy_gradient_loss  | -0.0054       |
|    agent/train/value_loss            | 0.0873        |
--------------------------------------------------------
-----------------------------------------------------
| mean/                                  |          |
|    agent/rollout/ep_rew_wrapped_mean   | 2.67     |
|    agent/time/fps                      | 173      |
|    agent/time/iterations               | 1        |
|    agent/time/time_elapsed             | 0        |
|    agent/time/total_timesteps          | 32       |
|    agent/train/approx_kl               | 0.000191 |
|    agent/train/clip_fraction           | 0        |
|    agent/train/clip_range              | 0.2      |
|    agent/train/entropy_loss            | -2.64    |
|    agent/train/explained_variance      | -0.263   |
|    agent/train/learning_rate           | 0.00025  |
|    agent/train/loss                    | -0.0303  |
|    agent/train/n_updates               | 8        |
|    agent/train/policy_gradient_loss    | -0.00671 |
```

(continues on next page)

(continued from previous page)

```
|    agent/train/value_loss              | 0.0423   |
|    preferences/entropy                 | 0.656    |
|    reward/epoch-0/train/accuracy       | 0.8      |
|    reward/epoch-0/train/gt_reward_loss | 0.679    |
|    reward/epoch-0/train/loss           | 0.574    |
|    reward/epoch-1/train/accuracy       | 0.8      |
|    reward/epoch-1/train/gt_reward_loss | 0.679    |
|    reward/epoch-1/train/loss           | 0.564    |
|    reward/epoch-2/train/accuracy       | 0.8      |
|    reward/epoch-2/train/gt_reward_loss | 0.679    |
|    reward/epoch-2/train/loss           | 0.553    |
| reward/                                |          |
|    final/train/accuracy                | 0.8      |
|    final/train/gt_reward_loss          | 0.679    |
|    final/train/loss                    | 0.553    |
-----------------------------------------------------
Collecting 10 fragments (100 transitions)
Requested 100 transitions but only 0 in buffer. Sampling 100 additional transitions.
Creating fragment pairs
Gathering preferences
Dataset now contains 15 comparisons
Training agent for 8 timesteps
--------------------------------------------------------
| raw/                                 |               |
|    agent/rollout/ep_rew_wrapped_mean | 2.7           |
|    agent/time/fps                    | 160           |
|    agent/time/iterations             | 1             |
|    agent/time/time_elapsed           | 0             |
|    agent/time/total_timesteps        | 48            |
|    agent/train/approx_kl             | 0.00019109249 |
|    agent/train/clip_fraction         | 0             |
|    agent/train/clip_range            | 0.2           |
|    agent/train/entropy_loss          | -2.64         |
|    agent/train/explained_variance    | -0.263        |
|    agent/train/learning_rate         | 0.00025       |
|    agent/train/loss                  | -0.0303       |
|    agent/train/n_updates             | 8             |
|    agent/train/policy_gradient_loss  | -0.00671      |
|    agent/train/value_loss            | 0.0423        |
--------------------------------------------------------
-----------------------------------------------------
| mean/                                  |          |
|    agent/rollout/ep_rew_wrapped_mean   | 2.7      |
|    agent/time/fps                      | 160      |
|    agent/time/iterations               | 1        |
|    agent/time/time_elapsed             | 0        |
|    agent/time/total_timesteps          | 48       |
|    agent/train/approx_kl               | 0.000199 |
|    agent/train/clip_fraction           | 0        |
|    agent/train/clip_range              | 0.2      |
|    agent/train/entropy_loss            | -2.64    |
|    agent/train/explained_variance      | -1.03    |
|    agent/train/learning_rate           | 0.00025  |
|    agent/train/loss                    | -0.0102  |
|    agent/train/n_updates               | 12       |
|    agent/train/policy_gradient_loss    | -0.00536 |
|    agent/train/value_loss              | 0.105    |
```

(continues on next page)

(continued from previous page)

```
|    preferences/entropy                | 0.693    |
|    reward/epoch-0/train/accuracy      | 0.733    |
|    reward/epoch-0/train/gt_reward_loss | 0.684    |
|    reward/epoch-0/train/loss          | 0.604    |
|    reward/epoch-1/train/accuracy      | 0.733    |
|    reward/epoch-1/train/gt_reward_loss | 0.684    |
|    reward/epoch-1/train/loss          | 0.594    |
|    reward/epoch-2/train/accuracy      | 0.733    |
|    reward/epoch-2/train/gt_reward_loss | 0.684    |
|    reward/epoch-2/train/loss          | 0.582    |
| reward/                               |          |
|    final/train/accuracy               | 0.733    |
|    final/train/gt_reward_loss         | 0.684    |
|    final/train/loss                   | 0.582    |
-----------------------------------------------------
```

```
{'reward_loss': 0.5816755890846252, 'reward_accuracy': 0.7333333492279053}
```

We can now wrap the environment with the learned reward model, shaped by the policy's learned value function. Note that if we were training this for real, we would want to normalize the output of the reward net as well as the value function, to ensure their values are on the same scale. To do this, use the `NormalizedRewardNet` class from `src/imitation/rewards/reward_nets.py` on `reward_net`, and modify the potential to add a `RunningNorm` module from `src/imitation/util/networks.py`.

```python
from imitation.rewards.reward_nets import ShapedRewardNet, cnn_transpose
from imitation.rewards.reward_wrapper import RewardVecEnvWrapper


def value_potential(state):
    state_ = cnn_transpose(state)
    return agent.policy.predict_values(state_)


shaped_reward_net = ShapedRewardNet(
    base=reward_net,
    potential=value_potential,
    discount_factor=0.99,
)

# GOTCHA: When using the NormalizedRewardNet wrapper, you should deactivate updating
# during evaluation by passing update_stats=False to the predict_processed method.
learned_reward_venv = RewardVecEnvWrapper(venv, shaped_reward_net.predict_processed)
```

Next, we train an agent that sees only the shaped, learned reward.

```python
learner = PPO(
    policy=CnnPolicy,
    env=learned_reward_venv,
    seed=0,
    batch_size=64,
    ent_coef=0.0,
    learning_rate=0.0003,
    n_epochs=10,
    n_steps=64,
)
learner.learn(1000)
```

```
<stable_baselines3.ppo.ppo.PPO at 0x7f2d5e23d9d0>
```

We now evaluate the learner using the original reward.

```
from stable_baselines3.common.evaluation import evaluate_policy

reward, _ = evaluate_policy(learner.policy, venv, 10)
print(reward)
```

```
0.4
```

### 2.24.1 Generating rollouts

When generating rollouts in image environments, be sure to use the agent's get_env() function rather than using the original environment.

The learner re-arranges the observations space to put the channel environment in the first dimension, and get_env() will correctly provide a wrapped environment doing this.

```
from imitation.data import rollout

rollouts = rollout.rollout(
    learner,
    # Note that passing venv instead of agent.get_env()
    # here would fail.
    learner.get_env(),
    rollout.make_sample_until(min_timesteps=None, min_episodes=3),
    rng=rng,
)
```

download this notebook here

## 2.25 Learn a Reward Function using Maximum Conditional Entropy Inverse Reinforcement Learning

Here, we're going to take a tabular environment with a pre-defined reward function, Cliffworld, and solve for the optimal policy. We then generate demonstrations from this policy, and use them to learn an approximation to the true reward function with MCE IRL. Finally, we directly compare the learned reward to the ground-truth reward (which we have access to in this example).

Cliffworld is a POMDP, and its "observations" consist of the (partial) observations proper and the (full) hidden environment state. We use `DictExtractWrapper` to extract only the hidden states from the environment, turning it into a fully observable MDP to make computing the optimal policy easy.

```
from functools import partial

from seals import base_envs
from seals.diagnostics.cliff_world import CliffWorldEnv
from stable_baselines3.common.vec_env import DummyVecEnv

import numpy as np

```

(continues on next page)

(continued from previous page)

```
from imitation.algorithms.mce_irl import (
    MCEIRL,
    mce_occupancy_measures,
    mce_partition_fh,
    TabularPolicy,
)
from imitation.data import rollout
from imitation.rewards import reward_nets

env_creator = partial(CliffWorldEnv, height=4, horizon=40, width=7, use_xy_obs=True)
env_single = env_creator()

state_env_creator = lambda: base_envs.ExposePOMDPStateWrapper(env_creator())

# This is just a vectorized environment because `generate_trajectories` expects one
state_venv = DummyVecEnv([state_env_creator] * 4)
```

Then we derive an expert policy using Bellman backups. We analytically compute the occupancy measures, and also sample some expert trajectories.

```
_, _, pi = mce_partition_fh(env_single)

_, om = mce_occupancy_measures(env_single, pi=pi)

rng = np.random.default_rng()
expert = TabularPolicy(
    state_space=env_single.state_space,
    action_space=env_single.action_space,
    pi=pi,
    rng=rng,
)

expert_trajs = rollout.generate_trajectories(
    policy=expert,
    venv=state_venv,
    sample_until=rollout.make_min_timesteps(5000),
    rng=rng,
)

print("Expert stats: ", rollout.rollout_stats(expert_trajs))
```

```
Expert stats:  {'n_traj': 128, 'return_min': 296.0, 'return_mean': 325.109375,
↪'return_std': 9.347655433817348, 'return_max': 334.0, 'len_min': 40, 'len_mean': 40.
↪0, 'len_std': 0.0, 'len_max': 40}
```

### 2.25.1 Training the reward function

The true reward here is not linear in the reduced feature space (i.e $(x, y)$ coordinates). Finding an appropriate linear reward is impossible, but an MLP should Just Work™.

```python
import matplotlib.pyplot as plt
import torch as th


def train_mce_irl(demos, hidden_sizes, lr=0.01, **kwargs):
    reward_net = reward_nets.BasicRewardNet(
        env_single.observation_space,
        env_single.action_space,
        hid_sizes=hidden_sizes,
        use_action=False,
        use_done=False,
        use_next_state=False,
    )

    mce_irl = MCEIRL(
        demos,
        env_single,
        reward_net,
        log_interval=250,
        optimizer_kwargs=dict(lr=lr),
        rng=rng,
    )
    occ_measure = mce_irl.train(**kwargs)

    imitation_trajs = rollout.generate_trajectories(
        policy=mce_irl.policy,
        venv=state_venv,
        sample_until=rollout.make_min_timesteps(5000),
        rng=rng,
    )
    print("Imitation stats: ", rollout.rollout_stats(imitation_trajs))

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    env_single.draw_value_vec(occ_measure)
    plt.title("Occupancy for learned reward")
    plt.xlabel("Gridworld x-coordinate")
    plt.ylabel("Gridworld y-coordinate")
    plt.subplot(1, 2, 2)
    _, true_occ_measure = mce_occupancy_measures(env_single)
    env_single.draw_value_vec(true_occ_measure)
    plt.title("Occupancy for true reward")
    plt.xlabel("Gridworld x-coordinate")
    plt.ylabel("Gridworld y-coordinate")
    plt.show()

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    env_single.draw_value_vec(
        reward_net(th.as_tensor(env_single.observation_matrix), None, None, None)
        .detach()
        .numpy()
    )
```

(continues on next page)

(continued from previous page)

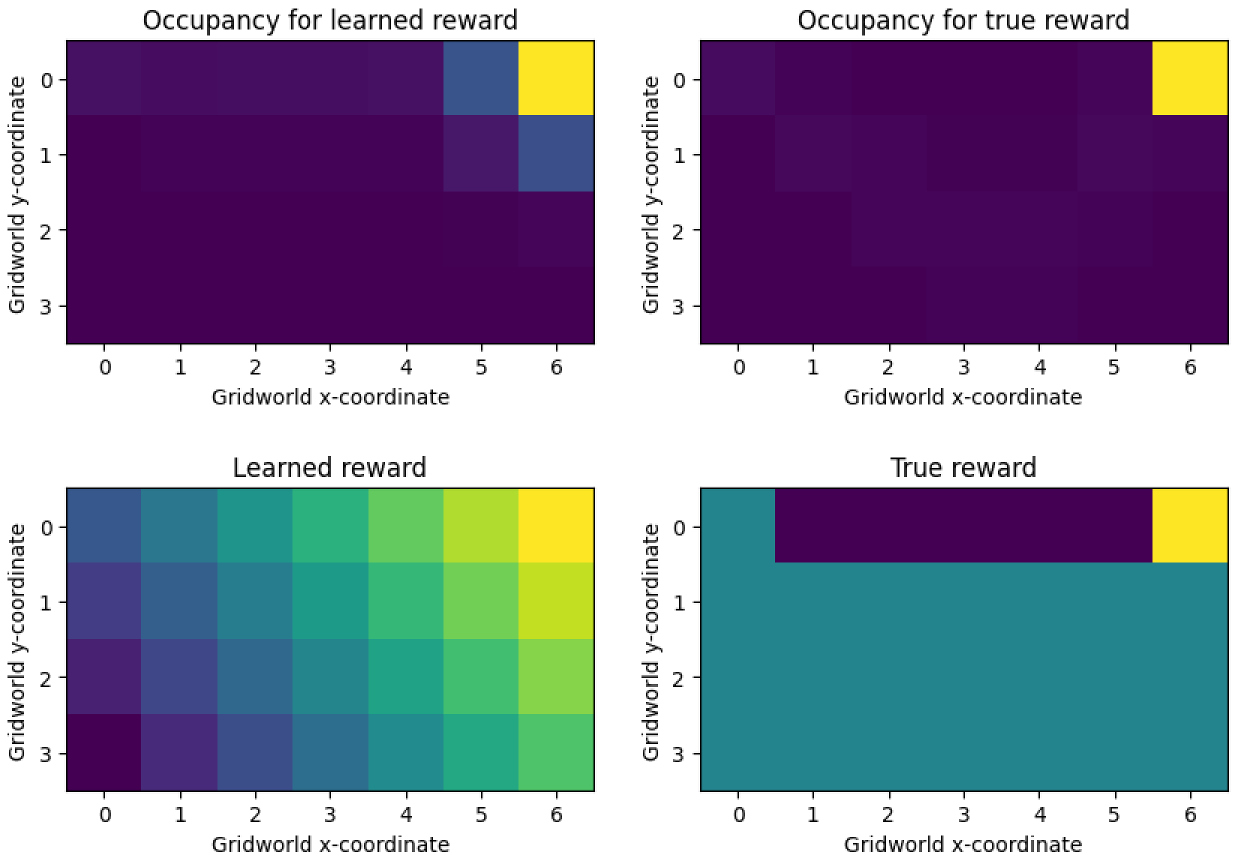
```
    plt.title("Learned reward")
    plt.xlabel("Gridworld x-coordinate")
    plt.ylabel("Gridworld y-coordinate")
    plt.subplot(1, 2, 2)
    env_single.draw_value_vec(env_single.reward_matrix)
    plt.title("True reward")
    plt.xlabel("Gridworld x-coordinate")
    plt.ylabel("Gridworld y-coordinate")
    plt.show()

    return mce_irl
```

As you can see, a linear reward model cannot fit the data. Even though we're training the model on analytically computed occupancy measures for the optimal policy, the resulting reward and occupancy frequencies diverge sharply.

```
train_mce_irl(om, hidden_sizes=[])
```

```
--------------------------
| grad_norm   | 19.9     |
| iteration   | 0        |
| linf_delta  | 31.1     |
| weight_norm | 0.883    |
--------------------------
--------------------------
| grad_norm   | 3.27    |
| iteration   | 250     |
| linf_delta  | 17.9    |
| weight_norm | 2.56    |
--------------------------
--------------------------
| grad_norm   | 1.95     |
| iteration   | 500      |
| linf_delta  | 14.7     |
| weight_norm | 4.14     |
--------------------------
--------------------------
| grad_norm   | 1.42     |
| iteration   | 750      |
| linf_delta  | 12.9     |
| weight_norm | 5.59     |
--------------------------
Imitation stats:  {'n_traj': 128, 'return_min': -12.0, 'return_mean': 115.9375,
↪'return_std': 40.625336537067604, 'return_max': 225.0, 'len_min': 40, 'len_mean':␣
↪40.0, 'len_std': 0.0, 'len_max': 40}
```



```
<imitation.algorithms.mce_irl.MCEIRL at 0x7f0d18132c10>
```

Now, let's try using a very simple nonlinear reward model: an MLP with a single hidden layer. We first train it on the analytically computed occupancy measures. This should give a very precise result.

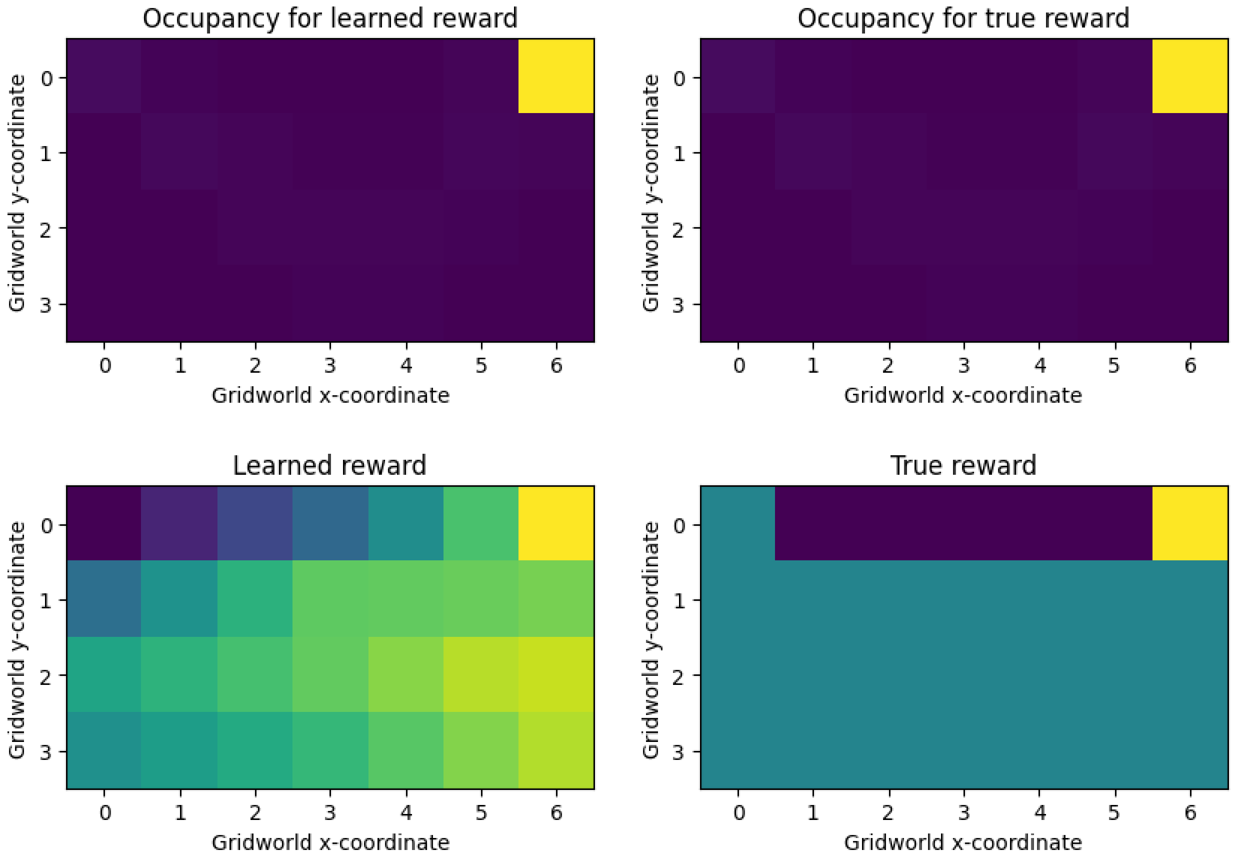
```
train_mce_irl(om, hidden_sizes=[256])
```

```
--------------------------
| grad_norm   | 72.7     |
| iteration   | 0        |
| linf_delta  | 30.8     |
| weight_norm | 11.4     |
--------------------------
--------------------------
| grad_norm   | 0.256    |
| iteration   | 250      |
| linf_delta  | 0.202    |
| weight_norm | 17.5     |
--------------------------
--------------------------
| grad_norm   | 0.28     |
| iteration   | 500      |
| linf_delta  | 0.128    |
| weight_norm | 19.7     |
--------------------------
--------------------------
```

(continues on next page)

(continued from previous page)

```
| grad_norm   | 0.19      |
| iteration   | 750       |
| linf_delta  | 0.0468    |
| weight_norm | 22        |
--------------------------
Imitation stats:  {'n_traj': 128, 'return_min': 289.0, 'return_mean': 324.7890625,
↪'return_std': 9.140100812961187, 'return_max': 334.0, 'len_min': 40, 'len_mean': 40.
↪0, 'len_std': 0.0, 'len_max': 40}
```


Occupancy for learned reward

Occupancy for true reward

Learned reward

True reward

```
<imitation.algorithms.mce_irl.MCEIRL at 0x7f0d18132d30>
```

Then we train it on trajectories sampled from the expert. This gives a stochastic approximation to occupancy measure, so performance is a little worse. Using more expert trajectories should improve performance – try it!

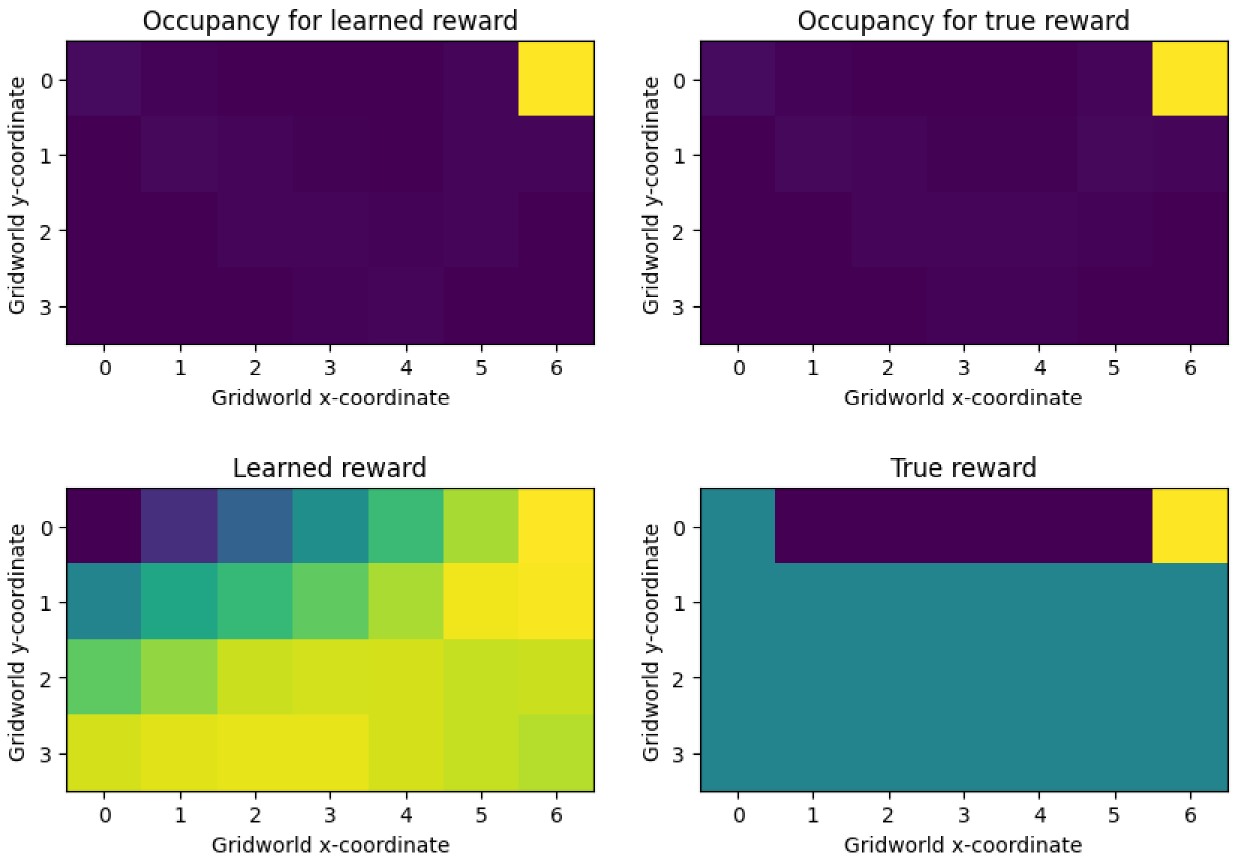
```
mce_irl_from_trajs = train_mce_irl(expert_trajs[0:10], hidden_sizes=[256])
```

```
--------------------------
| grad_norm   | 135       |
| iteration   | 0         |
| linf_delta  | 33.7      |
| weight_norm | 11.4      |
--------------------------
--------------------------
| grad_norm   | 2.21      |
| iteration   | 250       |
```

(continues on next page)

(continued from previous page)

```
| linf_delta  | 0.518    |
| weight_norm | 20.4     |
--------------------------
--------------------------
| grad_norm   | 5.83     |
| iteration   | 500      |
| linf_delta  | 0.511    |
| weight_norm | 34.8     |
--------------------------
--------------------------
| grad_norm   | 10.7     |
| iteration   | 750      |
| linf_delta  | 0.512    |
| weight_norm | 51.4     |
--------------------------
Imitation stats:  {'n_traj': 128, 'return_min': 296.0, 'return_mean': 325.7265625,
↪'return_std': 7.710296325926374, 'return_max': 334.0, 'len_min': 40, 'len_mean': 40.
↪0, 'len_std': 0.0, 'len_max': 40}
```



While the learned reward function is quite different from the true reward function, it induces a virtually identical occupancy measure over the states. In particular, states below the top row get almost the same reward as top-row states. This is because in Cliff World, there is an upward-blowing wind which will push the agent toward the top row with probability 0.3 at every timestep.

Even though the agent only gets reward in the top row squares, and maximum reward in the top righthand square, the reward model considers it to be almost as good to end up in one of the squares below the top rightmost square, since the wind will eventually blow the agent to the goal square.

download this notebook here

## 2.26 Learning a Reward Function using Kernel Density

This demo shows how to train a `Pendulum` agent (exciting!) with our simple density-based imitation learning baselines. `DensityTrainer` has a few interesting parameters, but the key ones are:

1. `density_type`: this governs whether density is measured on $(s, s')$ pairs (`db.STATE_STATE_DENSITY`), $(s, a)$ pairs (`db.STATE_ACTION_DENSITY`), or single states (`db.STATE_DENSITY`).
2. `is_stationary`: determines whether a separate density model is used for each time step $t$ (`False`), or the same model is used for transitions at all times (`True`).
3. `standardise_inputs`: if `True`, each dimension of the agent state vectors will be normalised to have zero mean and unit variance over the training dataset. This can be useful when not all elements of the demonstration vector are on the same scale, or when some elements have too wide a variation to be captured by the fixed kernel width (1 for Gaussian kernel).
4. `kernel`: changes the kernel used for non-parametric density estimation. `gaussian` and `exponential` are the best bets; see the sklearn docs for the rest.

```python
import pprint

from imitation.algorithms import density as db
from imitation.data import types
from imitation.util import util
```

```python
# Set FAST = False for longer training. Use True for testing and CI.
FAST = True

if FAST:
    N_VEC = 1
    N_TRAJECTORIES = 1
    N_ITERATIONS = 1
    N_RL_TRAIN_STEPS = 100

else:
    N_VEC = 8
    N_TRAJECTORIES = 10
    N_ITERATIONS = 10
    N_RL_TRAIN_STEPS = 100_000
```

```python
from imitation.policies.serialize import load_policy
from stable_baselines3.common.policies import ActorCriticPolicy
from stable_baselines3 import PPO
from imitation.data import rollout
from stable_baselines3.common.vec_env import DummyVecEnv
from stable_baselines3.common.evaluation import evaluate_policy
from imitation.data.wrappers import RolloutInfoWrapper
import gymnasium as gym
import numpy as np

SEED = 42

rng = np.random.default_rng(seed=SEED)
env_name = "Pendulum-v1"
```

(continues on next page)

(continued from previous page)

```
rollout_env = DummyVecEnv(
    [lambda: RolloutInfoWrapper(gym.make(env_name)) for _ in range(N_VEC)]
)
expert = load_policy(
    "ppo-huggingface",
    organization="HumanCompatibleAI",
    env_name=env_name,
    venv=rollout_env,
)
rollouts = rollout.rollout(
    expert,
    rollout_env,
    rollout.make_sample_until(min_timesteps=2000, min_episodes=57),
    rng=rng,
)

env = util.make_vec_env(env_name, n_envs=N_VEC, rng=rng)


imitation_trainer = PPO(
    ActorCriticPolicy, env, learning_rate=3e-4, gamma=0.95, ent_coef=1e-4, n_
↪steps=2048
)
density_trainer = db.DensityAlgorithm(
    venv=env,
    rng=rng,
    demonstrations=rollouts,
    rl_algo=imitation_trainer,
    density_type=db.DensityType.STATE_ACTION_DENSITY,
    is_stationary=True,
    kernel="gaussian",
    kernel_bandwidth=0.4,  # found using divination & some palm reading
    standardise_inputs=True,
)
density_trainer.train()
```

```
# evaluate the expert
expert_rewards, _ = evaluate_policy(expert, env, 100, return_episode_rewards=True)

# evaluate the learner before training
learner_rewards_before_training, _ = evaluate_policy(
    density_trainer.policy, env, 100, return_episode_rewards=True
)
```

```
def print_stats(density_trainer, n_trajectories, epoch=""):
    stats = density_trainer.test_policy(n_trajectories=n_trajectories)
    print("True reward function stats:")
    pprint.pprint(stats)
    stats_im = density_trainer.test_policy(
        true_reward=False,
        n_trajectories=n_trajectories,
    )
    print(f"Imitation reward function stats, epoch {epoch}:")
    pprint.pprint(stats_im)
```

(continues on next page)

(continued from previous page)

```
novice_stats = density_trainer.test_policy(n_trajectories=N_TRAJECTORIES)
print("Stats before training:")
print_stats(density_trainer, 1)

print("Starting the training!")
for i in range(N_ITERATIONS):
    density_trainer.train_policy(N_RL_TRAIN_STEPS)
    print_stats(density_trainer, 1, epoch=str(i))
```

```
Stats before training:
True reward function stats:
{'len_max': 200,
 'len_mean': 200.0,
 'len_min': 200,
 'len_std': 0.0,
 'monitor_return_len': 1,
 'monitor_return_max': -1493.001723,
 'monitor_return_mean': -1493.001723,
 'monitor_return_min': -1493.001723,
 'monitor_return_std': 0.0,
 'n_traj': 1,
 'return_max': -1493.001723766327,
 'return_mean': -1493.001723766327,
 'return_min': -1493.001723766327,
 'return_std': 0.0}
Imitation reward function stats, epoch :
{'len_max': 200,
 'len_mean': 200.0,
 'len_min': 200,
 'len_std': 0.0,
 'monitor_return_len': 1,
 'monitor_return_max': -1749.369344,
 'monitor_return_mean': -1749.369344,
 'monitor_return_min': -1749.369344,
 'monitor_return_std': 0.0,
 'n_traj': 1,
 'return_max': -2212.1580998897552,
 'return_mean': -2212.1580998897552,
 'return_min': -2212.1580998897552,
 'return_std': 0.0}
Starting the training!
True reward function stats:
{'len_max': 200,
 'len_mean': 200.0,
 'len_min': 200,
 'len_std': 0.0,
 'monitor_return_len': 1,
 'monitor_return_max': -908.535786,
 'monitor_return_mean': -908.535786,
 'monitor_return_min': -908.535786,
 'monitor_return_std': 0.0,
 'n_traj': 1,
 'return_max': -908.5357865467668,
 'return_mean': -908.5357865467668,
 'return_min': -908.5357865467668,
 'return_std': 0.0}
```

(continues on next page)

(continued from previous page)

```
Imitation reward function stats, epoch 0:
{'len_max': 200,
 'len_mean': 200.0,
 'len_min': 200,
 'len_std': 0.0,
 'monitor_return_len': 1,
 'monitor_return_max': -855.283381,
 'monitor_return_mean': -855.283381,
 'monitor_return_min': -855.283381,
 'monitor_return_std': 0.0,
 'n_traj': 1,
 'return_max': -2239.7023117542267,
 'return_mean': -2239.7023117542267,
 'return_min': -2239.7023117542267,
 'return_std': 0.0}
```

```
# evaluate the learner after training
learner_rewards_after_training, _ = evaluate_policy(
    density_trainer.policy, env, 100, return_episode_rewards=True
)
```

Here are the final results. If you set `FAST = False` in one of the initial cells, you should see that performance after training approaches that of an expert.

```
print("Mean expert reward:", np.mean(expert_rewards))
print("Mean reward before training:", np.mean(learner_rewards_before_training))
print("Mean reward after training:", np.mean(learner_rewards_after_training))
```

```
Mean expert reward: -212.67203443999998
Mean reward before training: -1235.5171938299998
Mean reward after training: -1145.53928535
```

download this notebook here

## 2.27 Train an Agent using Soft Q Imitation Learning

Soft Q Imitation Learning (SQIL) is a simple algorithm that can be used to clone expert behavior. It's fundamentally a modification of the DQN algorithm. At each training step, whenever we sample a batch of data from the replay buffer, we also sample a batch of expert data. Expert demonstrations are assigned a reward of 1, while the agent's own transitions are assigned a reward of 0. This approach encourages the agent to imitate the expert's behavior, but also to avoid unfamiliar states.

In this tutorial we will use the `imitation` library to train an agent using SQIL.

First, we need some expert trajectories in our environment (`seals/CartPole-v0`). Note that you can use other environments, but the action space must be discrete for this algorithm.

```
import datasets
from stable_baselines3.common.vec_env import DummyVecEnv

from imitation.data import huggingface_utils

# Download some expert trajectories from the HuggingFace Datasets Hub.
dataset = datasets.load_dataset("HumanCompatibleAI/ppo-CartPole-v1")
```

(continues on next page)

(continued from previous page)

```
# Convert the dataset to a format usable by the imitation library.
expert_trajectories = huggingface_utils.TrajectoryDatasetSequence(dataset["train"])
```

Let's quickly check if the expert is any good. We usually should be able to reach a reward of 500, which is the maximum achievable value.

```
from imitation.data import rollout

trajectory_stats = rollout.rollout_stats(expert_trajectories)

print(
    f"We have {trajectory_stats['n_traj']} trajectories."
    f"The average length of each trajectory is {trajectory_stats['len_mean']}."
    f"The average return of each trajectory is {trajectory_stats['return_mean']}."
)
```

```
We have 100 trajectories.The average length of each trajectory is 500.0.The average␣
↪return of each trajectory is 500.0.
```

After we collected our expert trajectories, it's time to set up our imitation algorithm.

```
from imitation.algorithms import sqil
import gymnasium as gym

venv = DummyVecEnv([lambda: gym.make("CartPole-v1")])
sqil_trainer = sqil.SQIL(
    venv=venv,
    demonstrations=expert_trajectories,
    policy="MlpPolicy",
)
```

As you can see the untrained policy only gets poor rewards:

```
from stable_baselines3.common.evaluation import evaluate_policy

reward_before_training, _ = evaluate_policy(sqil_trainer.policy, venv, 10)
print(f"Reward before training: {reward_before_training}")
```

```
Reward before training: 8.9
```

After training, we can match the rewards of the expert (500):

```
sqil_trainer.train(
    total_timesteps=1_000,
)  # Note: set to 1_000_000 to obtain good results
reward_after_training, _ = evaluate_policy(sqil_trainer.policy, venv, 10)
print(f"Reward after training: {reward_after_training}")
```

```
Reward after training: 9.2
```

download this notebook here

## 2.28 Train an Agent using Soft Q Imitation Learning with SAC

In the previous tutorial, we used Soft Q Imitation Learning (SQIL) on top of the DQN base algorithm. In fact, SQIL can be combined with any off-policy algorithm from `stable_baselines3`. Here, we train a Pendulum agent using SQIL + SAC.

First, we need some expert trajectories in our environment (`Pendulum-v1`). Note that you can use other environments, but the action space must be continuous.

```
import datasets
from imitation.data import huggingface_utils

# Download some expert trajectories from the HuggingFace Datasets Hub.
dataset = datasets.load_dataset("HumanCompatibleAI/ppo-Pendulum-v1")

# Convert the dataset to a format usable by the imitation library.
expert_trajectories = huggingface_utils.TrajectoryDatasetSequence(dataset["train"])
```

Let's quickly check if the expert is any good.

```
from imitation.data import rollout

trajectory_stats = rollout.rollout_stats(expert_trajectories)

print(
    f"We have {trajectory_stats['n_traj']} trajectories. "
    f"The average length of each trajectory is {trajectory_stats['len_mean']}. "
    f"The average return of each trajectory is {trajectory_stats['return_mean']}."
)
```

```
We have 200 trajectories. The average length of each trajectory is 200.0. The average␣
↪return of each trajectory is -205.22814517737746.
```

After we collected our expert trajectories, it's time to set up our imitation algorithm.

```
from imitation.algorithms import sqil
from imitation.util.util import make_vec_env
import numpy as np
from stable_baselines3 import sac

SEED = 42

venv = make_vec_env(
    "Pendulum-v1",
    rng=np.random.default_rng(seed=SEED),
)

sqil_trainer = sqil.SQIL(
    venv=venv,
    demonstrations=expert_trajectories,
    policy="MlpPolicy",
    rl_algo_class=sac.SAC,
    rl_kwargs=dict(seed=SEED),
)
```

As you can see the untrained policy only gets poor rewards (< 0):

```
from stable_baselines3.common.evaluation import evaluate_policy

reward_before_training, _ = evaluate_policy(sqil_trainer.policy, venv, 100)
print(f"Reward before training: {reward_before_training}")
```

```
Reward before training: -1386.1941136000003
```

After training, we can observe that agent is quite improved (> 1000), although it does not reach the expert performance in this case.

```
sqil_trainer.train(
    total_timesteps=1000,
)  # Note: set to 300_000 to obtain good results
reward_after_training, _ = evaluate_policy(sqil_trainer.policy, venv, 100)
print(f"Reward after training: {reward_after_training}")
```

```
Reward after training: -1217.9038355900002
```

download this notebook here

## 2.29 Reliably compare algorithm performance

Did we actually match the expert performance or was it just luck? Did this hyperparameter change actually improve the performance of our algorithm? These are questions that we need to answer when we want to compare the performance of different algorithms or hyperparameters.

imitation provides some tools to help you answer these questions. For demonstration purposes, we will use Behavior Cloning on the CartPole-v1 environment. We will compare different variants of the trained algorithm, and also compare it with a more sophisticated algorithm, DAgger.

We will start by training a good (but not perfect) expert.

```
import gymnasium as gym
from stable_baselines3 import PPO
from stable_baselines3.ppo import MlpPolicy

env = gym.make("CartPole-v1")
expert = PPO(
    policy=MlpPolicy,
    env=env,
    seed=0,
    batch_size=64,
    ent_coef=0.0,
    learning_rate=0.0003,
    n_epochs=10,
    n_steps=64,
)
expert.learn(10_000)  # set to 100_000 for better performance
```

```
<stable_baselines3.ppo.ppo.PPO at 0x7f10670d4970>
```

For comparison, let's also train a not-quite-expert.

```
not_expert = PPO(
    policy=MlpPolicy,
    env=env,
    seed=0,
    batch_size=64,
    ent_coef=0.0,
    learning_rate=0.0003,
    n_epochs=10,
    n_steps=64,
)

not_expert.learn(1_000)  # set to 10_000 for slightly better performance
```

```
<stable_baselines3.ppo.ppo.PPO at 0x7f10603b5dc0>
```

So are they any good? Let's quickly get a point estimate of their performance.

```
from stable_baselines3.common.evaluation import evaluate_policy

env.reset(seed=0)

expert_reward, _ = evaluate_policy(expert, env, 1)
not_expert_reward, _ = evaluate_policy(not_expert, env, 1)

print(f"Expert reward: {expert_reward:.2f}")
print(f"Not expert reward: {not_expert_reward:.2f}")
```

```
Expert reward: 147.00
Not expert reward: 71.00
```

But wait! We only ran the evaluation once. What if we got lucky? Let's run the evaluation a few more times and see what happens.

```
expert_reward, _ = evaluate_policy(expert, env, 10)
not_expert_reward, _ = evaluate_policy(not_expert, env, 10)

print(f"Expert reward: {expert_reward:.2f}")
print(f"Not expert reward: {not_expert_reward:.2f}")
```

```
Expert reward: 143.90
Not expert reward: 83.40
```

Seems a bit more robust now, but how certain are we? Fortunately, `imitation` provides us with tools to answer this.

We will perform a permutation test using the `is_significant_reward_improvement` function. We want to be very certain – let's set the bar high and require a p-value of 0.001.

```
from imitation.testing.reward_improvement import is_significant_reward_improvement

expert_rewards, _ = evaluate_policy(expert, env, 10, return_episode_rewards=True)
not_expert_rewards, _ = evaluate_policy(
    not_expert, env, 10, return_episode_rewards=True
)

significant = is_significant_reward_improvement(
    not_expert_rewards, expert_rewards, 0.001
```

(continues on next page)

(continued from previous page)

```
)

print(
    f"The expert is {'NOT ' if not significant else ''}significantly better than the␣
↪not-expert."
)
```

```
The expert is significantly better than the not-expert.
```

Huh, turns out we set the bar too high. We could lower our standards, but that's for cowards. Instead, we can collect more data and try again.

```
from imitation.testing.reward_improvement import is_significant_reward_improvement

expert_rewards, _ = evaluate_policy(expert, env, 100, return_episode_rewards=True)
not_expert_rewards, _ = evaluate_policy(
    not_expert, env, 100, return_episode_rewards=True
)

significant = is_significant_reward_improvement(
    not_expert_rewards, expert_rewards, 0.001
)

print(
    f"The expert is {'NOT ' if not significant else ''}significantly better than the␣
↪not-expert."
)
```

```
The expert is significantly better than the not-expert.
```

Here we go! We can now be 99.9% confident that the expert is better than the not-expert – in this specific case, with these specific trained models. It might still be an extraordinary stroke of luck, or a conspiracy to make us choose the wrong algorithm, but outside of that, we can be pretty sure our data's correct.

We can use the same principle to with imitation learning algorithms. Let's train a behavior cloning algorithm and see how it compares to the expert. This time, we can lower the bar to the standard "scientific" threshold of 0.05.

Like in the first tutorial, we will start by collecting some expert data. But to spice it up, let's also get some data from the not-quite-expert.

```
from imitation.data import rollout
from imitation.data.wrappers import RolloutInfoWrapper
from stable_baselines3.common.vec_env import DummyVecEnv
import numpy as np

rng = np.random.default_rng()
expert_rollouts = rollout.rollout(
    expert,
    DummyVecEnv([lambda: RolloutInfoWrapper(env)]),
    rollout.make_sample_until(min_timesteps=None, min_episodes=50),
    rng=rng,
)
expert_transitions = rollout.flatten_trajectories(expert_rollouts)


not_expert_rollouts = rollout.rollout(
```

(continues on next page)

(continued from previous page)

```
    not_expert,
    DummyVecEnv([lambda: RolloutInfoWrapper(env)]),
    rollout.make_sample_until(min_timesteps=None, min_episodes=50),
    rng=rng,
)
not_expert_transitions = rollout.flatten_trajectories(not_expert_rollouts)
```

Let's try cloning an expert and a non-expert, and see how they compare.

```
from imitation.algorithms import bc

expert_bc_trainer = bc.BC(
    observation_space=env.observation_space,
    action_space=env.action_space,
    demonstrations=expert_transitions,
    rng=rng,
)

not_expert_bc_trainer = bc.BC(
    observation_space=env.observation_space,
    action_space=env.action_space,
    demonstrations=not_expert_transitions,
    rng=rng,
)
```

```
expert_bc_trainer.train(n_epochs=2)
not_expert_bc_trainer.train(n_epochs=2)
```

```
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -0.000693 |
|    entropy        | 0.693     |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 72.5      |
|    loss           | 0.693     |
|    neglogp        | 0.693     |
|    prob_true_act  | 0.5       |
|    samples_so_far | 32        |
---------------------------------
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -0.000693 |
|    entropy        | 0.693     |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 72.5      |
|    loss           | 0.693     |
|    neglogp        | 0.693     |
|    prob_true_act  | 0.5       |
|    samples_so_far | 32        |
---------------------------------
```

```
bc_expert_rewards, _ = evaluate_policy(
    expert_bc_trainer.policy, env, 10, return_episode_rewards=True
)
bc_not_expert_rewards, _ = evaluate_policy(
    not_expert_bc_trainer.policy, env, 10, return_episode_rewards=True
)
significant = is_significant_reward_improvement(
    bc_not_expert_rewards, bc_expert_rewards, 0.05
)
print(f"Cloned expert rewards: {bc_expert_rewards}")
print(f"Cloned not-expert rewards: {bc_not_expert_rewards}")

print(
    f"Cloned expert is {'NOT ' if not significant else ''}significantly better than␣
↪the cloned not-expert."
)
```

```
Cloned expert rewards: [121.0, 133.0, 153.0, 140.0, 123.0, 109.0, 139.0, 116.0, 108.0,
↪ 134.0]
Cloned not-expert rewards: [47.0, 94.0, 84.0, 54.0, 77.0, 88.0, 122.0, 253.0, 105.0,␣
↪62.0]
Cloned expert is NOT significantly better than the cloned not-expert.
```

How about comparing the expert clone to the expert itself?

```
bc_clone_rewards, _ = evaluate_policy(
    expert_bc_trainer.policy, env, 10, return_episode_rewards=True
)

expert_rewards, _ = evaluate_policy(expert, env, 10, return_episode_rewards=True)

significant = is_significant_reward_improvement(bc_clone_rewards, expert_rewards, 0.
↪05)

print(f"Cloned expert rewards: {bc_clone_rewards}")
print(f"Expert rewards: {expert_rewards}")

print(
    f"Expert is {'NOT ' if not significant else ''}significantly better than the␣
↪cloned expert."
)
```

```
Cloned expert rewards: [106.0, 132.0, 155.0, 144.0, 131.0, 116.0, 114.0, 129.0, 117.0,
↪ 115.0]
Expert rewards: [140.0, 132.0, 154.0, 126.0, 121.0, 138.0, 175.0, 132.0, 132.0, 139.0]
Expert is significantly better than the cloned expert.
```

Turns out the expert is significantly better than the clone – again, in this case. Note, however, that this is not proof that the clone is as good as the expert – there's a subtle difference between the two claims in the context of hypothesis testing.

Note: if you changed the duration of the training at the beginning of this tutorial, you might get different results. While this might break the narrative in this tutorial, it's a good learning opportunity.

When comparing the performance of two agents, algorithms, hyperparameter sets, always remember the scope of what you're testing. In this tutorial, we have one instance of an expert – but RL training is famously unstable, so another training run with another random seed would likely produce a slightly different result. So ideally, we would like to repeat this procedure several times, training the same agent with different random seeds, and then compare the average performance of the two agents.

Even then, this is just on one environment, with one algorithm. So be wary of generalizing your results too much.

We can also use the same method to compare different algorithms. While CartPole is pretty easy, we can make it more difficult by decreasing the number of episodes in our dataset, and generating them with a suboptimal policy:

```
rollouts = rollout.rollout(
    expert,
    DummyVecEnv([lambda: RolloutInfoWrapper(env)]),
    rollout.make_sample_until(min_timesteps=None, min_episodes=1),
    rng=rng,
)
transitions = rollout.flatten_trajectories(rollouts)
```

Let's try training a behavior cloning algorithm on this dataset.

Note that for DAgger, we have to cheat a little bit – it's allowed to use the expert policy to generate additional data. For the purposes of this tutorial, we'll stick with this to avoid spending hours training an expert for a more complex environment.

So while this little experiment isn't definitive proof that DAgger is better than BC, you can use the same method to compare any two algorithms.

```
from imitation.algorithms.dagger import SimpleDAggerTrainer
import tempfile

bc_trainer = bc.BC(
    observation_space=env.observation_space,
    action_space=env.action_space,
    demonstrations=transitions,
    rng=rng,
)

bc_trainer.train(n_epochs=1)


with tempfile.TemporaryDirectory(prefix="dagger_example_") as tmpdir:
    print(tmpdir)
    dagger_bc_trainer = bc.BC(
        observation_space=env.observation_space,
        action_space=env.action_space,
        rng=np.random.default_rng(),
    )
    dagger_trainer = SimpleDAggerTrainer(
        venv=DummyVecEnv([lambda: RolloutInfoWrapper(env)]),
        scratch_dir=tmpdir,
        expert_policy=expert,
        bc_trainer=dagger_bc_trainer,
        rng=np.random.default_rng(),
    )

    dagger_trainer.train(5000)
```

```
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -0.000693 |
|    entropy        | 0.693     |
|    epoch          | 0         |
```

(continues on next page)

(continued from previous page)

```
|    l2_loss        | 0         |
|    l2_norm        | 72.5      |
|    loss           | 0.693     |
|    neglogp        | 0.694     |
|    prob_true_act  | 0.5       |
|    samples_so_far | 32        |
---------------------------------
/tmp/dagger_example_amfg9vpt
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -0.000693 |
|    entropy        | 0.693     |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 72.5      |
|    loss           | 0.692     |
|    neglogp        | 0.693     |
|    prob_true_act  | 0.5       |
|    samples_so_far | 32        |
| rollout/          |           |
|    return_max     | 43        |
|    return_mean    | 25.8      |
|    return_min     | 17        |
|    return_std     | 10.2      |
---------------------------------
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -0.000654 |
|    entropy        | 0.654     |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 75.8      |
|    loss           | 0.519     |
|    neglogp        | 0.52      |
|    prob_true_act  | 0.602     |
|    samples_so_far | 32        |
| rollout/          |           |
|    return_max     | 59        |
|    return_mean    | 48.4      |
|    return_min     | 42        |
|    return_std     | 6.18      |
---------------------------------
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -0.000227 |
|    entropy        | 0.227     |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 88.6      |
|    loss           | 0.106     |
|    neglogp        | 0.106     |
```

(continues on next page)

(continued from previous page)

```
|    prob_true_act  | 0.91      |
|    samples_so_far | 32        |
| rollout/          |           |
|    return_max     | 216       |
|    return_mean    | 169       |
|    return_min     | 111       |
|    return_std     | 39.6      |
---------------------------------
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -9.87e-05 |
|    entropy        | 0.0987    |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 99.6      |
|    loss           | 0.0455    |
|    neglogp        | 0.0456    |
|    prob_true_act  | 0.961     |
|    samples_so_far | 32        |
| rollout/          |           |
|    return_max     | 177       |
|    return_mean    | 144       |
|    return_min     | 128       |
|    return_std     | 17.2      |
---------------------------------
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -7.96e-05 |
|    entropy        | 0.0796    |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 109       |
|    loss           | 0.102     |
|    neglogp        | 0.102     |
|    prob_true_act  | 0.936     |
|    samples_so_far | 32        |
| rollout/          |           |
|    return_max     | 129       |
|    return_mean    | 124       |
|    return_min     | 116       |
|    return_std     | 5.16      |
---------------------------------
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -7.47e-05 |
|    entropy        | 0.0747    |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 118       |
|    loss           | 0.0752    |
|    neglogp        | 0.0753    |
```

(continues on next page)

(continued from previous page)

```
|    prob_true_act  | 0.951      |
|    samples_so_far | 32         |
| rollout/          |            |
|    return_max     | 136        |
|    return_mean    | 129        |
|    return_min     | 122        |
|    return_std     | 5.55       |
---------------------------------
---------------------------------
| batch_size        | 32         |
| bc/               |            |
|    batch          | 0          |
|    ent_loss       | -3.24e-05  |
|    entropy        | 0.0324     |
|    epoch          | 0          |
|    l2_loss        | 0          |
|    l2_norm        | 127        |
|    loss           | 0.0106     |
|    neglogp        | 0.0107     |
|    prob_true_act  | 0.99       |
|    samples_so_far | 32         |
| rollout/          |            |
|    return_max     | 132        |
|    return_mean    | 128        |
|    return_min     | 126        |
|    return_std     | 2.53       |
---------------------------------
---------------------------------
| batch_size        | 32         |
| bc/               |            |
|    batch          | 0          |
|    ent_loss       | -6.06e-06  |
|    entropy        | 0.00606    |
|    epoch          | 0          |
|    l2_loss        | 0          |
|    l2_norm        | 136        |
|    loss           | 0.00124    |
|    neglogp        | 0.00124    |
|    prob_true_act  | 0.999      |
|    samples_so_far | 32         |
| rollout/          |            |
|    return_max     | 149        |
|    return_mean    | 135        |
|    return_min     | 125        |
|    return_std     | 9.77       |
---------------------------------
---------------------------------
| batch_size        | 32         |
| bc/               |            |
|    batch          | 500        |
|    ent_loss       | -4.32e-05  |
|    entropy        | 0.0432     |
|    epoch          | 3          |
|    l2_loss        | 0          |
|    l2_norm        | 144        |
|    loss           | 0.0192     |
|    neglogp        | 0.0192     |
```

(continues on next page)

(continued from previous page)

```
|    prob_true_act  | 0.983     |
|    samples_so_far | 16032     |
| rollout/          |           |
|    return_max     | 141       |
|    return_mean    | 127       |
|    return_min     | 119       |
|    return_std     | 8.36      |
---------------------------------
---------------------------------
| batch_size        | 32        |
| bc/               |           |
|    batch          | 0         |
|    ent_loss       | -1.25e-06 |
|    entropy        | 0.00125   |
|    epoch          | 0         |
|    l2_loss        | 0         |
|    l2_norm        | 145       |
|    loss           | 0.000184  |
|    neglogp        | 0.000186  |
|    prob_true_act  | 1         |
|    samples_so_far | 32        |
| rollout/          |           |
|    return_max     | 195       |
|    return_mean    | 152       |
|    return_min     | 128       |
|    return_std     | 29.1      |
---------------------------------
--------------------------------
| batch_size        | 32       |
| bc/               |          |
|    batch          | 500      |
|    ent_loss       | -3.6e-05 |
|    entropy        | 0.036    |
|    epoch          | 3        |
|    l2_loss        | 0        |
|    l2_norm        | 153      |
|    loss           | 0.0146   |
|    neglogp        | 0.0146   |
|    prob_true_act  | 0.987    |
|    samples_so_far | 16032    |
| rollout/          |          |
|    return_max     | 173      |
|    return_mean    | 144      |
|    return_min     | 121      |
|    return_std     | 19       |
--------------------------------
```

After training both BC and DAgger, let's compare their performances again! We expect DAgger to be better – after all, it's a more advanced algorithm. But is it significantly better?

```
bc_rewards, _ = evaluate_policy(bc_trainer.policy, env, 10, return_episode_
↪rewards=True)
dagger_rewards, _ = evaluate_policy(
    dagger_trainer.policy, env, 10, return_episode_rewards=True
)

significant = is_significant_reward_improvement(bc_rewards, dagger_rewards, 0.05)
```

```
print(f"BC rewards: {bc_rewards}")
print(f"DAgger rewards: {dagger_rewards}")

print(
    f"Our DAgger agent is {'NOT ' if not significant else ''}significantly better␣
↪than BC."
)
```

```
BC rewards: [82.0, 69.0, 68.0, 115.0, 98.0, 80.0, 97.0, 118.0, 62.0, 78.0]
DAgger rewards: [125.0, 134.0, 118.0, 141.0, 148.0, 126.0, 177.0, 125.0, 121.0, 130.0]
Our DAgger agent is significantly better than BC.
```

If you increased the number of training iterations for the expert (in the first cell of the tutorial), you should see that DAgger indeed performs better than BC. If you didn't, you likely see the opposite result. Yet another reason to be careful when interpreting results!

Finally, let's take a moment, to remember the limitations of this experiment. We're comparing two algorithms on one environment, with one dataset. We're also using a suboptimal expert policy, which might not be the best choice for BC. If you want to convince yourself that DAgger is better than BC, you should pick out a more complex environment, you should run this experiment several times, with different random seeds and perform some hyperparameter optimization to make sure we're not just using unlucky hyperparameters. At the end, we would also need to run the same hypothesis test across average returns of several independent runs.

But now you have all the pieces of the puzzle to do that!

download this notebook here

## 2.30 Train Behavior Cloning in a Custom Environment

You can use `imitation` to train a policy (and, for many imitation learning algorithm, learn rewards) in a custom environment.

### 2.30.1 Step 1: Define the environment

We will use a simple ObservationMatching environment as an example. The premise is simple – the agent receives a vector of observations, and must output a vector of actions that matches the observations as closely as possible.

If you have your own environment that you'd like to use, you can replace the code below with your own environment. Make sure it complies with the standard Gym API, and that the observation and action spaces are specified correctly.

```
from typing import Dict, Optional
from typing import Any
import numpy as np
import gymnasium as gym

from gymnasium.spaces import Box


class ObservationMatchingEnv(gym.Env):
    def __init__(self, num_options: int = 2):
        self.state = None
        self.num_options = num_options
        self.observation_space = Box(0, 1, shape=(num_options,))
        self.action_space = Box(0, 1, shape=(num_options,))
```

(continues on next page)

(continued from previous page)

```
    def reset(self, seed: int = None, options: Optional[Dict[str, Any]] = None):
        super().reset(seed=seed, options=options)
        self.state = self.observation_space.sample()
        return self.state, {}

    def step(self, action):
        reward = -np.abs(self.state - action).mean()
        self.state = self.observation_space.sample()
        return self.state, reward, False, False, {}
```

## 2.30.2 Step 2: create the environment

From here, we have two options:

- Add the environment to the gym registry, and use it with existing utilities (e.g. `make`)
- Use the environment directly

You only need to execute the cells in step 2a, or step 2b to proceed.

At the end of these steps, we want to have:

- `env`: a single environment that we can use for training an expert with SB3
- `venv`: a vectorized environment where each individual environment is wrapped in `RolloutInfoWrapper`, that we can use for collecting rollouts with `imitation`

### Step 2a (recommended): add the environment to the gym registry

The standard approach is adding the environment to the gym registry.

```
gym.register(
    id="custom/ObservationMatching-v0",
    entry_point=ObservationMatchingEnv,  # This can also be the path to the class, e.
↪g. `observation_matching:ObservationMatchingEnv`
    max_episode_steps=500,
)
```

After registering, you can create an environment is `gym.make(env_id)` which automatically handles the `TimeLimit` wrapper.

To create a vectorized env, you can use the `make_vec_env` helper function (Option A), or create it directly (Options B1 and B2)

```
from gymnasium.wrappers import TimeLimit
from imitation.data import rollout
from imitation.data.wrappers import RolloutInfoWrapper
from imitation.util.util import make_vec_env
from stable_baselines3.common.vec_env import DummyVecEnv, SubprocVecEnv

# Create a single environment for training an expert with SB3
env = gym.make("custom/ObservationMatching-v0")


# Create a vectorized environment for training with `imitation`
```

(continues on next page)

(continued from previous page)

```
# Option A: use the `make_vec_env` helper function - make sure to pass `post_
↪wrappers=[lambda env, _: RolloutInfoWrapper(env)]`
venv = make_vec_env(
    "custom/ObservationMatching-v0",
    rng=np.random.default_rng(),
    n_envs=4,
    post_wrappers=[lambda env, _: RolloutInfoWrapper(env)],
)


# Option B1: use a custom env creator, and create VecEnv directly
# def _make_env():
#     """Helper function to create a single environment. Put any logic here, but make␣
↪sure to return a RolloutInfoWrapper."""
#     _env = gym.make("custom/ObservationMatching-v0")
#     _env = RolloutInfoWrapper(_env)
#     return _env
#
# venv = DummyVecEnv([_make_env for _ in range(4)])
#
# # Option B2: we can also use a parallel VecEnv implementation
# venv = SubprocVecEnv([_make_env for _ in range(4)])
```

### Step 2b: directly use the environment

Alternatively, we can directly initialize the environment by instantiating the class we created earlier, and handle all the additional logic ourselves.

```
from gymnasium.wrappers import TimeLimit
from imitation.data import rollout
from imitation.data.wrappers import RolloutInfoWrapper
from stable_baselines3.common.vec_env import DummyVecEnv
import numpy as np

# Create a single environment for training with SB3
env = ObservationMatchingEnv()
env = TimeLimit(env, max_episode_steps=500)

# Create a vectorized environment for training with `imitation`


# Option A: use a helper function to create multiple environments
def _make_env():
    """Helper function to create a single environment. Put any logic here, but make␣
↪sure to return a RolloutInfoWrapper."""
    _env = ObservationMatchingEnv()
    _env = TimeLimit(_env, max_episode_steps=500)
    _env = RolloutInfoWrapper(_env)
    return _env


venv = DummyVecEnv([_make_env for _ in range(4)])
```

(continues on next page)

(continued from previous page)

```
# Option B: use a single environment
# env = FixedHorizonCartPoleEnv()
# venv = DummyVecEnv([lambda: RolloutInfoWrapper(env)])  # Wrap a single environment --- only useful for simple testing like this

# Option C: use multiple environments
# venv = DummyVecEnv([lambda: RolloutInfoWrapper(ObservationMatchingEnv()) for _ in range(4)])  # Wrap multiple environments
```

### 2.30.3 Step 3: Training

And now we're just about done! Whether you used step 2a or 2b, your environment should now be ready to use with SB3 and `imitation`.

For the sake of completeness, we'll train a BC model, the same way as in the first tutorial, but with our custom environment.

Keep in mind that while we're using BC in this tutorial, you can just as easily use any of the other algorithms with the environment prepared in this way.

```
from stable_baselines3 import PPO
from stable_baselines3.ppo import MlpPolicy
from stable_baselines3.common.evaluation import evaluate_policy
from gymnasium.wrappers import TimeLimit

expert = PPO(
    policy=MlpPolicy,
    env=env,
    seed=0,
    batch_size=64,
    ent_coef=0.0,
    learning_rate=0.0003,
    n_epochs=10,
    n_steps=64,
)

reward, _ = evaluate_policy(expert, env, 10)
print(f"Reward before training: {reward}")


# Note: if you followed step 2a, i.e. registered the environment, you can use the environment name directly

# expert = PPO(
#     policy=MlpPolicy,
#     env="custom/ObservationMatching-v0",
#     seed=0,
#     batch_size=64,
#     ent_coef=0.0,
#     learning_rate=0.0003,
#     n_epochs=10,
#     n_steps=64,
# )
expert.learn(10_000)  # Note: set to 100000 to train a proficient expert
reward, _ = evaluate_policy(expert, expert.get_env(), 10)
print(f"Expert reward: {reward}")
```

```
Reward before training: -247.16096657663584
Expert reward: -106.15597290000001
```

```
rng = np.random.default_rng()
rollouts = rollout.rollout(
    expert,
    venv,
    rollout.make_sample_until(min_timesteps=None, min_episodes=50),
    rng=rng,
)
transitions = rollout.flatten_trajectories(rollouts)
```

```
from imitation.algorithms import bc

bc_trainer = bc.BC(
    observation_space=env.observation_space,
    action_space=env.action_space,
    demonstrations=transitions,
    rng=rng,
)
```

As before, the untrained policy only gets poor rewards:

```
reward_before_training, _ = evaluate_policy(bc_trainer.policy, env, 10)
print(f"Reward before training: {reward_before_training}")
```

```
Reward before training: -251.080459844321
```

After training, we can get much closer to the expert's performance:

```
bc_trainer.train(n_epochs=1)
reward_after_training, _ = evaluate_policy(bc_trainer.policy, env, 10)
print(f"Reward after training: {reward_after_training}")
```

```
--------------------------------
| batch_size        | 32       |
| bc/               |          |
|    batch          | 0        |
|    ent_loss       | -0.00284 |
|    entropy        | 2.84     |
|    epoch          | 0        |
|    l2_loss        | 0        |
|    l2_norm        | 68.5     |
|    loss           | 2.22     |
|    neglogp        | 2.22     |
|    prob_true_act  | 0.113    |
|    samples_so_far | 32       |
--------------------------------
--------------------------------
| batch_size        | 32       |
| bc/               |          |
|    batch          | 500      |
|    ent_loss       | -0.00182 |
|    entropy        | 1.82     |
|    epoch          | 0        |
|    l2_loss        | 0        |
```

(continues on next page)

(continued from previous page)

```
|    l2_norm        | 74.5     |
|    loss           | 1.09     |
|    neglogp        | 1.09     |
|    prob_true_act  | 0.349    |
|    samples_so_far | 16032    |
--------------------------------
Reward after training: -37.813750486366914
```

CHAPTER

THREE

# API REFERENCE

|  |  |
|---|---|
| *imitation* | imitation: implementations of imitation and reward learning algorithms. |

## 3.1 imitation

imitation: implementations of imitation and reward learning algorithms.

### Modules

|  |  |
|---|---|
| *imitation.algorithms* | Implementations of imitation and reward learning algorithms. |
| *imitation.data* | Modules handling environment data. |
| *imitation.policies* | Classes defining policies and methods to manipulate them (e.g. |
| *imitation.regularization* | Implements a variety of regularization techniques for NN weights. |
| *imitation.rewards* | Reward models: neural network modules, serialization, preprocessing, etc. |
| *imitation.scripts* | Command-line scripts. |
| *imitation.testing* | Helper methods for unit tests. |
| *imitation.util* | General utility functions: e.g. |

### 3.1.1 imitation.algorithms

Implementations of imitation and reward learning algorithms.

### Modules

| | |
|---|---|
| *imitation.algorithms.adversarial* | Adversarial imitation learning algorithms, AIRL and GAIL. |
| *imitation.algorithms.base* | Module of base classes and helper methods for imitation learning algorithms. |
| *imitation.algorithms.bc* | Behavioural Cloning (BC). |
| *imitation.algorithms.dagger* | DAgger (https://arxiv.org/pdf/1011.0686.pdf). |
| *imitation.algorithms.density* | Density-based baselines for imitation learning. |
| *imitation.algorithms.mce_irl* | Finite-horizon tabular Maximum Causal Entropy IRL. |
| *imitation.algorithms.preference_comparisons* | Learning reward models using preference comparisons. |
| *imitation.algorithms.sqil* | Soft Q Imitation Learning (SQIL) (https://arxiv.org/abs/1905.11108). |

### imitation.algorithms.adversarial

Adversarial imitation learning algorithms, AIRL and GAIL.

### Modules

| | |
|---|---|
| *imitation.algorithms.adversarial.airl* | Adversarial Inverse Reinforcement Learning (AIRL). |
| *imitation.algorithms.adversarial.common* | Core code for adversarial imitation learning, shared between GAIL and AIRL. |
| *imitation.algorithms.adversarial.gail* | Generative Adversarial Imitation Learning (GAIL). |

### imitation.algorithms.adversarial.airl

Adversarial Inverse Reinforcement Learning (AIRL).

### Classes

| | |
|---|---|
| *AIRL*(*, demonstrations, demo_batch_size, ...) | Adversarial Inverse Reinforcement Learning (AIRL). |

**class** `imitation.algorithms.adversarial.airl.`**AIRL**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, **kwargs*)

Bases: *AdversarialTrainer*

Adversarial Inverse Reinforcement Learning (AIRL).

**__init__**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, **kwargs*)

Builds an AIRL trainer.

**Parameters**

- **demonstrations** (`Union[Iterable[`*Trajectory*`], Iterable[`*TransitionMapping*`],` *TransitionsMinimal*`]`) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).

- **demo_batch_size** (int) – The number of samples in each batch of expert data. The discriminator batch size is twice this number because each discriminator batch contains a generator sample for every expert sample.
- **venv** (VecEnv) – The vectorized environment to train in.
- **gen_algo** (BaseAlgorithm) – The generator RL algorithm that is trained to maximize discriminator confusion. Environment and logger will be set to *venv* and *custom_logger*.
- **reward_net** (*RewardNet*) – Reward network; used as part of AIRL discriminator.
- ****kwargs** – Passed through to *AdversarialTrainer.__init__*.

**Raises**

**TypeError** – If *gen_algo.policy* does not have an *evaluate_actions* attribute (present in *ActorCriticPolicy*), needed to compute log-probability of actions.

**logits_expert_is_high**(*state*, *action*, *next_state*, *done*, *log_policy_act_prob=None*)

Compute the discriminator's logits for each state-action sample.

In Fu's AIRL paper (https://arxiv.org/pdf/1710.11248.pdf), the discriminator output was given as

$$D_\theta(s,a) = \frac{\exp r_\theta(s,a)}{\exp r_\theta(s,a) + \pi(a|s)}$$

with a high value corresponding to the expert and a low value corresponding to the generator.

In other words, the discriminator output is the probability that the action is taken by the expert rather than the generator.

The logit of the above is given as

$$\operatorname{logit}(D_\theta(s,a)) = r_\theta(s,a) - \log \pi(a|s)$$

which is what is returned by this function.

**Parameters**

- **state** (Tensor) – The state of the environment at the time of the action.
- **action** (Tensor) – The action taken by the expert or generator.
- **next_state** (Tensor) – The state of the environment after the action.
- **done** (Tensor) – whether a *terminal state* (as defined under the MDP of the task) has been reached.
- **log_policy_act_prob** (Optional[Tensor]) – The log probability of the action taken by the generator, $\log \pi(a|s)$.

**Return type**

Tensor

**Returns**

The logits of the discriminator for each state-action sample.

**Raises**

**TypeError** – If *log_policy_act_prob* is None.

**property reward_test: *RewardNet***

Returns the unshaped version of reward network used for testing.

**Return type**

*RewardNet*

**property reward_train: *RewardNet***

Reward used to train generator policy.

**Return type**

*RewardNet*

**venv: VecEnv**

The original vectorized environment.

**venv_train: VecEnv**

Like *self.venv*, but wrapped with train reward unless in debug mode.

If *debug_use_ground_truth=True* was passed into the initializer then *self.venv_train* is the same as *self.venv*.

**venv_wrapped: VecEnvWrapper**

## imitation.algorithms.adversarial.common

Core code for adversarial imitation learning, shared between GAIL and AIRL.

### Functions

| | |
|---|---|
| *compute_train_stats*(...) | Train statistics for GAIL/AIRL discriminator. |

### Classes

| | |
|---|---|
| *AdversarialTrainer*(*, demonstrations, ...[, ...]) | Base class for adversarial imitation learning algorithms like GAIL and AIRL. |

**class** imitation.algorithms.adversarial.common.**AdversarialTrainer**(*, demonstrations, demo_batch_size, venv, gen_algo, reward_net, demo_minibatch_size=None, n_disc_updates_per_round=2, log_dir='output/', disc_opt_cls=<class 'torch.optim.adam.Adam'>, disc_opt_kwargs=None, gen_train_timesteps=None, gen_replay_buffer_capacity=None, custom_logger=None, init_tensorboard=False, init_tensorboard_graph=False, debug_use_ground_truth=False, allow_variable_horizon=False)

Bases: DemonstrationAlgorithm[Transitions]

Base class for adversarial imitation learning algorithms like GAIL and AIRL.

**__init__**(*, demonstrations, demo_batch_size, venv, gen_algo, reward_net, demo_minibatch_size=None, n_disc_updates_per_round=2, log_dir='output/', disc_opt_cls=<class 'torch.optim.adam.Adam'>, disc_opt_kwargs=None, gen_train_timesteps=None, gen_replay_buffer_capacity=None, custom_logger=None, init_tensorboard=False, init_tensorboard_graph=False, debug_use_ground_truth=False, allow_variable_horizon=False)

Builds AdversarialTrainer.

**Parameters**

- **demonstrations** (Union[Iterable[Trajectory], Iterable[TransitionMapping], TransitionsMinimal]) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **demo_batch_size** (int) – The number of samples in each batch of expert data. The discriminator batch size is twice this number because each discriminator batch contains a generator sample for every expert sample.
- **venv** (VecEnv) – The vectorized environment to train in.
- **gen_algo** (BaseAlgorithm) – The generator RL algorithm that is trained to maximize discriminator confusion. Environment and logger will be set to *venv* and *custom_logger*.
- **reward_net** (RewardNet) – a Torch module that takes an observation, action and next

observation tensors as input and computes a reward signal.

- **demo_minibatch_size** (Optional[int]) – size of minibatch to calculate gradients over. The gradients are accumulated until the entire batch is processed before making an optimization step. This is useful in GPU training to reduce memory usage, since fewer examples are loaded into memory at once, facilitating training with larger batch sizes, but is generally slower. Must be a factor of *demo_batch_size*. Optional, defaults to *demo_batch_size*.
- **n_disc_updates_per_round** (int) – The number of discriminator updates after each round of generator updates in AdversarialTrainer.learn().
- **log_dir** (Union[str, bytes, PathLike]) – Directory to store TensorBoard logs, plots, etc. in.
- **disc_opt_cls** (Type[Optimizer]) – The optimizer for discriminator training.
- **disc_opt_kwargs** (Optional[Mapping]) – Parameters for discriminator training.
- **gen_train_timesteps** (Optional[int]) – The number of steps to train the generator policy for each iteration. If None, then defaults to the batch size (for on-policy) or number of environments (for off-policy).
- **gen_replay_buffer_capacity** (Optional[int]) – The capacity of the generator replay buffer (the number of obs-action-obs samples from the generator that can be stored). By default this is equal to *gen_train_timesteps*, meaning that we sample only from the most recent batch of generator samples.
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **init_tensorboard** (bool) – If True, makes various discriminator TensorBoard summaries.
- **init_tensorboard_graph** (bool) – If both this and *init_tensorboard* are True, then write a Tensorboard graph summary to disk.
- **debug_use_ground_truth** (bool) – If True, use the ground truth reward for *self.train_env*. This disables the reward wrapping that would normally replace the environment reward with the learned reward. This is useful for sanity checking that the policy training is functional.
- **allow_variable_horizon** (bool) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/guide/variable_horizon.html before overriding this.

**Raises**

**ValueError** – if the batch size is not a multiple of the minibatch size.

**abstract logits_expert_is_high** (*state*, *action*, *next_state*, *done*, *log_policy_act_prob=None*)

Compute the discriminator's logits for each state-action sample.

A high value corresponds to predicting expert, and a low value corresponds to predicting generator.

**Parameters**

- **state** (Tensor) – state at time t, of shape *(batch_size,)* + *state_shape*.
- **action** (Tensor) – action taken at time t, of shape *(batch_size,)* + *action_shape*.
- **next_state** (Tensor) – state at time t+1, of shape *(batch_size,)* + *state_shape*.

- **done** (Tensor) – binary episode completion flag after action at time t, of shape *(batch_size,)*.
- **log_policy_act_prob** (Optional[Tensor]) – log probability of generator policy taking *action* at time t.

**Return type**
Tensor

**Returns**
Discriminator logits of shape *(batch_size,)*. A high output indicates an expert-like transition.

**property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**
BasePolicy

**abstract property reward_test:** ***RewardNet***

Reward used to train policy at "test" time after adversarial training.

**Return type**
*RewardNet*

**abstract property reward_train:** ***RewardNet***

Reward used to train generator policy.

**Return type**
*RewardNet*

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**
**demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**
None

**train**(*total_timesteps*, *callback=None*)

Alternates between training the generator and discriminator.

Every "round" consists of a call to *train_gen(self.gen_train_timesteps)*, a call to *train_disc*, and finally a call to *callback(round)*.

Training ends once an additional "round" would cause the number of transitions sampled from the environment to exceed *total_timesteps*.

**Parameters**

- **total_timesteps** (int) – An upper bound on the number of transitions to sample from the environment during training.
- **callback** (Optional[Callable[[int], None]]) – A function called at the end of every round which takes in a single argument, the round number. Round numbers are in *range(total_timesteps // self.gen_train_timesteps)*.

**Return type**
`None`

`train_disc`(**, expert_samples=None, gen_samples=None*)

Perform a single discriminator update, optionally using provided samples.

**Parameters**

- **expert_samples** (`Optional[Mapping]`) – Transition samples from the expert in dictionary form. If provided, must contain keys corresponding to every field of the *Transitions* dataclass except "infos". All corresponding values can be either NumPy arrays or Tensors. Extra keys are ignored. Must contain *self.demo_batch_size* samples. If this argument is not provided, then *self.demo_batch_size* expert samples from *self.demo_data_loader* are used by default.
- **gen_samples** (`Optional[Mapping]`) – Transition samples from the generator policy in same dictionary form as *expert_samples*. If provided, must contain exactly *self.demo_batch_size* samples. If not provided, then take *len(expert_samples)* samples from the generator replay buffer.

**Return type**
`Mapping[str, float]`

**Returns**
Statistics for discriminator (e.g. loss, accuracy).

`train_gen`(*total_timesteps=None, learn_kwargs=None*)

Trains the generator to maximize the discriminator loss.

After the end of training populates the generator replay buffer (used in discriminator training) with *self.disc_batch_size* transitions.

**Parameters**

- **total_timesteps** (`Optional[int]`) – The number of transitions to sample from *self.venv_train* during training. By default, *self.gen_train_timesteps*.
- **learn_kwargs** (`Optional[Mapping]`) – kwargs for the Stable Baselines *RLModel.learn()* method.

**Return type**
`None`

**venv: VecEnv**

The original vectorized environment.

**venv_train: VecEnv**

Like *self.venv*, but wrapped with train reward unless in debug mode.

If *debug_use_ground_truth=True* was passed into the initializer then *self.venv_train* is the same as *self.venv*.

**venv_wrapped: VecEnvWrapper**

`imitation.algorithms.adversarial.common.`**`compute_train_stats`**(*disc_logits_expert_is_high, labels_expert_is_one, disc_loss*)

Train statistics for GAIL/AIRL discriminator.

**Parameters**

- **disc_logits_expert_is_high** (`Tensor`) – discriminator logits produced by *AdversarialTrainer.logits_expert_is_high*.

- **labels_expert_is_one** (Tensor) – integer labels describing whether logit was for an expert (0) or generator (1) sample.
- **disc_loss** (Tensor) – final discriminator loss.

**Return type**

Mapping[str, float]

**Returns**

A mapping from statistic names to float values.

### imitation.algorithms.adversarial.gail

Generative Adversarial Imitation Learning (GAIL).

#### Classes

| | |
|---|---|
| *GAIL*(*, demonstrations, demo_batch_size, ...) | Generative Adversarial Imitation Learning (GAIL). |
| *RewardNetFromDiscriminatorLogit*(base) | Converts the discriminator logits raw value to a reward signal. |

**class** imitation.algorithms.adversarial.gail.**GAIL**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, **kwargs*)

Bases: *AdversarialTrainer*

Generative Adversarial Imitation Learning (GAIL).

**__init__**(**, demonstrations, demo_batch_size, venv, gen_algo, reward_net, **kwargs*)

Generative Adversarial Imitation Learning.

**Parameters**

- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **demo_batch_size** (int) – The number of samples in each batch of expert data. The discriminator batch size is twice this number because each discriminator batch contains a generator sample for every expert sample.
- **venv** (VecEnv) – The vectorized environment to train in.
- **gen_algo** (BaseAlgorithm) – The generator RL algorithm that is trained to maximize discriminator confusion. Environment and logger will be set to *venv* and *custom_logger*.
- **reward_net** (*RewardNet*) – a Torch module that takes an observation, action and next observation tensor as input, then computes the logits. Used as the GAIL discriminator.
- ****kwargs** – Passed through to *AdversarialTrainer.__init__*.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**logits_expert_is_high**(*state*, *action*, *next_state*, *done*, *log_policy_act_prob=None*)

Compute the discriminator's logits for each state-action sample.

**Parameters**

- **state** (Tensor) – The state of the environment at the time of the action.
- **action** (Tensor) – The action taken by the expert or generator.
- **next_state** (Tensor) – The state of the environment after the action.
- **done** (Tensor) – whether a *terminal state* (as defined under the MDP of the task) has been reached.
- **log_policy_act_prob** (Optional[Tensor]) – The log probability of the action taken by the generator, $\log P(a|s)$.

**Return type**

Tensor

**Returns**

The logits of the discriminator for each state-action sample.

**property reward_test:** ***RewardNet***

Reward used to train policy at "test" time after adversarial training.

**Return type**

*RewardNet*

**property reward_train:** ***RewardNet***

Reward used to train generator policy.

**Return type**

*RewardNet*

**venv: VecEnv**

The original vectorized environment.

**venv_train: VecEnv**

Like *self.venv*, but wrapped with train reward unless in debug mode.

If *debug_use_ground_truth=True* was passed into the initializer then *self.venv_train* is the same as *self.venv*.

**venv_wrapped: VecEnvWrapper**

**class** imitation.algorithms.adversarial.gail.**RewardNetFromDiscriminatorLogit**(*base*)

Bases: *RewardNet*

Converts the discriminator logits raw value to a reward signal.

Wrapper for reward network that takes in the logits of the discriminator probability distribution and outputs the corresponding reward for the GAIL algorithm.

Below is the derivation of the transformation that needs to be applied.

The GAIL paper defines the cost function of the generator as:

$$\log D$$

as shown on line 5 of Algorithm 1. In the paper, $D$ is the probability distribution learned by the discriminator, where $D(X) = 1$ if the trajectory comes from the generator, and $D(X) = 0$ if it comes from the expert. In this implementation, we have decided to use the opposite convention that $D(X) = 0$ if the trajectory comes from the generator, and $D(X) = 1$ if it comes from the expert. Therefore, the resulting cost function is:

$$\log(1 - D)$$

Since our algorithm trains using a reward function instead of a loss function, we need to invert the sign to get:

$$R = -\log(1 - D) = \log\frac{1}{1 - D}$$

Now, let $L$ be the output of our reward net, which gives us the logits of D ($L = \operatorname{logit} D$). We can write:

$$D = \operatorname{sigmoid} L = \frac{1}{1 + e^{-L}}$$

Since $1 - \operatorname{sigmoid}(L)$ is the same as $\operatorname{sigmoid}(-L)$, we can write:

$$R = -\log\operatorname{sigmoid}(-L)$$

which is a non-decreasing map from the logits of D to the reward.

**`__init__`**(*base*)

Builds LogSigmoidRewardNet to wrap *reward_net*.

**`forward`**(*state*, *action*, *next_state*, *done*)

Compute rewards for a batch of transitions and keep gradients.

**Return type**
`Tensor`

**`training: bool`**

### imitation.algorithms.base

Module of base classes and helper methods for imitation learning algorithms.

#### Functions

| | |
|---|---|
| *`make_data_loader`*(transitions, batch_size[, ...]) | Converts demonstration data to Torch data loader. |

#### Classes

| | |
|---|---|
| *`BaseImitationAlgorithm`*(*[, custom_logger, ...]) | Base class for all imitation learning algorithms. |
| *`DemonstrationAlgorithm`*(*, demonstrations[, ...]) | An algorithm that learns from demonstration: BC, IRL, etc. |

**`class`** `imitation.algorithms.base.`**`BaseImitationAlgorithm`**(*, *custom_logger=None*, *allow_variable_horizon=False*)

Bases: `ABC`

Base class for all imitation learning algorithms.

**__init__**(**, custom_logger=None, allow_variable_horizon=False*)

Creates an imitation learning algorithm.

**Parameters**

- **custom_logger** (`Optional[`*`HierarchicalLogger`*`]`) – Where to log to; if None (default), creates a new logger.
- **allow_variable_horizon** (`bool`) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/getting-started/variable-horizon.html before overriding this.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**property logger:** ***HierarchicalLogger***

**Return type**

*`HierarchicalLogger`*

**class** `imitation.algorithms.base.`**DemonstrationAlgorithm**(**, demonstrations, custom_logger=None, allow_variable_horizon=False*)

Bases: *`BaseImitationAlgorithm`*, `Generic[TransitionKind]`

An algorithm that learns from demonstration: BC, IRL, etc.

**__init__**(**, demonstrations, custom_logger=None, allow_variable_horizon=False*)

Creates an algorithm that learns from demonstrations.

**Parameters**

- **demonstrations** (`Union[Iterable[`*`Trajectory`*`], Iterable[`*`TransitionMapping`*`], `*`TransitionsMinimal`*`, None]`) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **custom_logger** (`Optional[`*`HierarchicalLogger`*`]`) – Where to log to; if None (default), creates a new logger.
- **allow_variable_horizon** (`bool`) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/getting-started/variable-horizon.html before overriding this.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**abstract property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**

`BasePolicy`

**abstract set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**

None

imitation.algorithms.base.**make_data_loader**(*transitions*, *batch_size*, *data_loader_kwargs=None*)

Converts demonstration data to Torch data loader.

**Parameters**

- **transitions** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **batch_size** (int) – The size of the batch to create. Does not change the batch size if *transitions* is already an iterable of transition batches.
- **data_loader_kwargs** (Optional[Mapping[str, Any]]) – Arguments to pass to *th_data.DataLoader*.

**Return type**

Iterable[*TransitionMapping*]

**Returns**

An iterable of transition batches.

**Raises**

- **ValueError** – if *transitions* is an iterable over transition batches with batch size not equal to *batch_size*; or if *transitions* is transitions or a sequence of trajectories with total timesteps less than *batch_size*.
- **TypeError** – if *transitions* is an unsupported type.

### imitation.algorithms.bc

Behavioural Cloning (BC).

Trains policy by applying supervised learning to a fixed dataset of (observation, action) pairs generated by some expert demonstrator.

### Functions

| | |
|---|---|
| *enumerate_batches*(batch_it) | Prepends batch stats before the batches of a batch iterator. |
| *reconstruct_policy*(policy_path[, device]) | Reconstruct a saved policy. |

### Classes

| | |
|---|---|
| *BC*(*, observation_space, action_space, rng) | Behavioral cloning (BC). |
| *BCLogger*(logger) | Utility class to help logging information relevant to Behavior Cloning. |
| *BCTrainingMetrics*(neglogp, entropy, ...) | Container for the different components of behavior cloning loss. |
| *BatchIteratorWithEpochEndCallback*(...) | Loops through batches from a batch loader and calls a callback after every epoch. |
| *BehaviorCloningLossCalculator*(ent_weight, ...) | Functor to compute the loss used in Behavior Cloning. |
| *RolloutStatsComputer*(venv, n_episodes) | Computes statistics about rollouts. |

**class** imitation.algorithms.bc.**BC**(**, observation_space, action_space, rng, policy=None, demonstrations=None, batch_size=32, minibatch_size=None, optimizer_cls=<class 'torch.optim.adam.Adam'>, optimizer_kwargs=None, ent_weight=0.001, l2_weight=0.0, device='auto', custom_logger=None*)

Bases: *DemonstrationAlgorithm*

Behavioral cloning (BC).

Recovers a policy via supervised learning from observation-action pairs.

**__init__**(**, observation_space, action_space, rng, policy=None, demonstrations=None, batch_size=32, minibatch_size=None, optimizer_cls=<class 'torch.optim.adam.Adam'>, optimizer_kwargs=None, ent_weight=0.001, l2_weight=0.0, device='auto', custom_logger=None*)

Builds BC.

**Parameters**

- **observation_space** (Space) – the observation space of the environment.
- **action_space** (Space) – the action space of the environment.
- **rng** (Generator) – the random state to use for the random number generator.
- **policy** (Optional[ActorCriticPolicy]) – a Stable Baselines3 policy; if unspecified, defaults to *FeedForward32Policy*.
- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*, None]) – Demonstrations from an expert (optional). Transitions expressed directly as a *types.TransitionsMinimal* object, a sequence of trajectories, or an iterable of transition batches (mappings from keywords to arrays containing observations, etc).
- **batch_size** (int) – The number of samples in each batch of expert data.
- **minibatch_size** (Optional[int]) – size of minibatch to calculate gradients over. The gradients are accumulated until *batch_size* examples are processed before making an

optimization step. This is useful in GPU training to reduce memory usage, since fewer examples are loaded into memory at once, facilitating training with larger batch sizes, but is generally slower. Must be a factor of *batch_size*. Optional, defaults to *batch_size*.

- **optimizer_cls** (`Type[Optimizer]`) – optimiser to use for supervised training.
- **optimizer_kwargs** (`Optional[Mapping[str, Any]]`) – keyword arguments, excluding learning rate and weight decay, for optimiser construction.
- **ent_weight** (`float`) – scaling applied to the policy's entropy regularization.
- **l2_weight** (`float`) – scaling applied to the policy's L2 regularization.
- **device** (`Union[str, device]`) – name/identity of device to place policy on.
- **custom_logger** (`Optional[HierarchicalLogger]`) – Where to log to; if None (default), creates a new logger.

**Raises**

**ValueError** – If *weight_decay* is specified in *optimizer_kwargs* (use the parameter *l2_weight* instead), or if the batch size is not a multiple of the minibatch size.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**property policy: ActorCriticPolicy**

Returns a policy imitating the demonstration data.

**Return type**

`ActorCriticPolicy`

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (`Union[Iterable[Trajectory], Iterable[TransitionMapping], TransitionsMinimal]`) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**

`None`

**train**(**, n_epochs=None, n_batches=None, on_epoch_end=None, on_batch_end=None, log_interval=500, log_rollouts_venv=None, log_rollouts_n_episodes=5, progress_bar=True, reset_tensorboard=False*)

Train with supervised learning for some number of epochs.

Here an 'epoch' is just a complete pass through the expert data loader, as set by *self.set_expert_data_loader()*. Note, that when you specify *n_batches* smaller than the number of batches in an epoch, the *on_epoch_end* callback will never be called.

**Parameters**

- **n_epochs** (`Optional[int]`) – Number of complete passes made through expert data before ending training. Provide exactly one of *n_epochs* and *n_batches*.
- **n_batches** (`Optional[int]`) – Number of batches loaded from dataset before ending training. Provide exactly one of *n_epochs* and *n_batches*.
- **on_epoch_end** (`Optional[Callable[[], None]]`) – Optional callback with no parameters to run at the end of each epoch.

- **on_batch_end** (Optional[Callable[[], None]]) – Optional callback with no parameters to run at the end of each batch.
- **log_interval** (int) – Log stats after every log_interval batches.
- **log_rollouts_venv** (Optional[VecEnv]) – If not None, then this VecEnv (whose observation and actions spaces must match *self.observation_space* and *self.action_space*) is used to generate rollout stats, including average return and average episode length. If None, then no rollouts are generated.
- **log_rollouts_n_episodes** (int) – Number of rollouts to generate when calculating rollout stats. Non-positive number disables rollouts.
- **progress_bar** (bool) – If True, then show a progress bar during training.
- **reset_tensorboard** (bool) – If True, then start plotting to Tensorboard from x=0 even if *.train()* logged to Tensorboard previously. Has no practical effect if *.train()* is being called for the first time.

**class** imitation.algorithms.bc.**BCLogger**(*logger*)

Bases: object

Utility class to help logging information relevant to Behavior Cloning.

**__init__**(*logger*)

Create new BC logger.

**Parameters**

**logger** (*HierarchicalLogger*) – The logger to feed all the information to.

**log_batch**(*batch_num*, *batch_size*, *num_samples_so_far*, *training_metrics*, *rollout_stats*)

**log_epoch**(*epoch_number*)

**reset_tensorboard_steps**()

**class** imitation.algorithms.bc.**BCTrainingMetrics**(*neglogp*, *entropy*, *ent_loss*, *prob_true_act*, *l2_norm*, *l2_loss*, *loss*)

Bases: object

Container for the different components of behavior cloning loss.

**__init__**(*neglogp*, *entropy*, *ent_loss*, *prob_true_act*, *l2_norm*, *l2_loss*, *loss*)

**ent_loss: Tensor**

**entropy: Optional[Tensor]**

**l2_loss: Tensor**

**l2_norm: Tensor**

**loss: Tensor**

**neglogp: Tensor**

**prob_true_act: Tensor**

class imitation.algorithms.bc.**BatchIteratorWithEpochEndCallback**(*batch_loader*, *n_epochs*, *n_batches*, *on_epoch_end*)

Bases: object

Loops through batches from a batch loader and calls a callback after every epoch.

Will throw an exception when an epoch contains no batches.

**__init__**(*batch_loader*, *n_epochs*, *n_batches*, *on_epoch_end*)

**batch_loader: Iterable[*TransitionMapping*]**

**n_batches: Optional[int]**

**n_epochs: Optional[int]**

**on_epoch_end: Optional[Callable[[int], None]]**

class imitation.algorithms.bc.**BehaviorCloningLossCalculator**(*ent_weight*, *l2_weight*)

Bases: object

Functor to compute the loss used in Behavior Cloning.

**__init__**(*ent_weight*, *l2_weight*)

**ent_weight: float**

**l2_weight: float**

class imitation.algorithms.bc.**RolloutStatsComputer**(*venv*, *n_episodes*)

Bases: object

Computes statistics about rollouts.

**Parameters**

- **venv** (Optional[VecEnv]) – The vectorized environment in which to compute the rollouts.
- **n_episodes** (int) – The number of episodes to base the statistics on.

**__init__**(*venv*, *n_episodes*)

**n_episodes: int**

**venv: Optional[VecEnv]**

imitation.algorithms.bc.**enumerate_batches**(*batch_it*)

Prepends batch stats before the batches of a batch iterator.

**Return type**

Iterable[Tuple[Tuple[int, int, int], *TransitionMapping*]]

imitation.algorithms.bc.**reconstruct_policy**(*policy_path*, *device='auto'*)

Reconstruct a saved policy.

**Parameters**

- **policy_path** (str) – path where *.save_policy()* has been run.
- **device** (Union[device, str]) – device on which to load the policy.

**Returns**
policy with reloaded weights.

**Return type**
policy

### imitation.algorithms.dagger

DAgger (https://arxiv.org/pdf/1011.0686.pdf).

Interactively trains policy by collecting some demonstrations, doing BC, collecting more demonstrations, doing BC again, etc. Initially the demonstrations just come from the expert's policy; over time, they shift to be drawn more and more from the imitator's policy.

#### Functions

| | |
|---|---|
| *reconstruct_trainer*(scratch_dir, venv[, ...]) | Reconstruct trainer from the latest snapshot in some working directory. |

#### Classes

| | |
|---|---|
| *BetaSchedule*() | Computes beta (% of time demonstration action used) from training round. |
| *DAggerTrainer*(*, venv, scratch_dir, rng[, ...]) | DAgger training class with low-level API suitable for interactive human feedback. |
| *ExponentialBetaSchedule*(decay_probability) | Exponentially decaying schedule for beta. |
| *InteractiveTrajectoryCollector*(venv, ...) | DAgger VecEnvWrapper for querying and saving expert actions. |
| *LinearBetaSchedule*(rampdown_rounds) | Linearly-decreasing schedule for beta. |
| *SimpleDAggerTrainer*(*, venv, scratch_dir, ...) | Simpler subclass of DAggerTrainer for training with synthetic feedback. |

#### Exceptions

| | |
|---|---|
| *NeedsDemosException* | Signals demos need to be collected for current round before continuing. |

**class** imitation.algorithms.dagger.**BetaSchedule**

Bases: ABC

Computes beta (% of time demonstration action used) from training round.

**class** imitation.algorithms.dagger.**DAggerTrainer**(*, *venv*, *scratch_dir*, *rng*, *beta_schedule=None*, *bc_trainer*, *custom_logger=None*)

Bases: *BaseImitationAlgorithm*

DAgger training class with low-level API suitable for interactive human feedback.

In essence, this is just BC with some helpers for incrementally resuming training and interpolating between demonstrator/learnt policies. Interaction proceeds in "rounds" in which the demonstrator first provides a fresh set of demonstrations, and then an underlying *BC* is invoked to fine-tune the policy on the entire set of demonstrations collected in all rounds so far. Demonstrations and policy/trainer checkpoints are stored in a directory with the following structure:

```
scratch-dir-name/
    checkpoint-001.pt
    checkpoint-002.pt
    …
    checkpoint-XYZ.pt
    checkpoint-latest.pt
    demos/
        round-000/
            demos_round_000_000.npz
            demos_round_000_001.npz
            …
        round-001/
            demos_round_001_000.npz
            …
        …
        round-XYZ/
            …
```

**DEFAULT_N_EPOCHS: int = 4**

The default number of BC training epochs in *extend_and_update*.

**__init__**(*, *venv*, *scratch_dir*, *rng*, *beta_schedule=None*, *bc_trainer*, *custom_logger=None*)

Builds DAggerTrainer.

**Parameters**

- **venv** (`VecEnv`) – Vectorized training environment.
- **scratch_dir** (`Union[str, bytes, PathLike]`) – Directory to use to store intermediate training information (e.g. for resuming training).
- **rng** (`Generator`) – random state for random number generation.
- **beta_schedule** (`Optional[Callable[[int], float]]`) – Provides a value of *beta* (the probability of taking expert action in any given state) at each round of training. If *None*, then *linear_beta_schedule* will be used instead.
- **bc_trainer** (`BC`) – A *BC* instance used to train the underlying policy.
- **custom_logger** (`Optional[HierarchicalLogger]`) – Where to log to; if None (default), creates a new logger.

**property batch_size: int**

**Return type**

`int`

**create_trajectory_collector**()

Create trajectory collector to extend current round's demonstration set.

**Return type**

`InteractiveTrajectoryCollector`

**Returns**

A collector configured with the appropriate beta, imitator policy, etc. for the current round. Refer to the documentation for *InteractiveTrajectoryCollector* to see how to use this.

**extend_and_update**(*bc_train_kwargs=None*)

Extend internal batch of data and train BC.

Specifically, this method will load new transitions (if necessary), train the model for a while, and advance the round counter. If there are no fresh demonstrations in the demonstration directory for the current round, then this will raise a *NeedsDemosException* instead of training or advancing the round counter. In that case, the user should call *.create_trajectory_collector()* and use the returned *InteractiveTrajectoryCollector* to produce a new set of demonstrations for the current interaction round.

**Parameters**

**bc_train_kwargs** (`Optional[Mapping[str, Any]]`) – Keyword arguments for calling *BC.train()*. If the *log_rollouts_venv* key is not provided, then it is set to *self.venv* by default. If neither of the *n_epochs* and *n_batches* keys are provided, then *n_epochs* is set to *self.DEFAULT_N_EPOCHS*.

**Return type**

`int`

**Returns**

New round number after advancing the round counter.

**property logger: *HierarchicalLogger***

Returns logger for this object.

**Return type**

*`HierarchicalLogger`*

**property policy: BasePolicy**

**Return type**

`BasePolicy`

**save_trainer**()

Create a snapshot of trainer in the scratch/working directory.

The created snapshot can be reloaded with *reconstruct_trainer()*. In addition to saving one copy of the policy in the trainer snapshot, this method saves a second copy of the policy in its own file. Having a second copy of the policy is convenient because it can be loaded on its own and passed to evaluation routines for other algorithms.

**Returns**

a path to one of the created *DAggerTrainer* checkpoints. policy_path: a path to one of the created *DAggerTrainer* policies.

**Return type**

checkpoint_path

**class** imitation.algorithms.dagger.**ExponentialBetaSchedule**(*decay_probability*)

Bases: *`BetaSchedule`*

Exponentially decaying schedule for beta.

**__init__**(*decay_probability*)

Builds ExponentialBetaSchedule.

**Parameters**

**decay_probability** (`float`) – the decay factor for beta.

**Raises**

**ValueError** – if *decay_probability* not within (0, 1].

**class** imitation.algorithms.dagger.**InteractiveTrajectoryCollector**(*venv*, *get_robot_acts*, *beta*, *save_dir*, *rng*)

Bases: `VecEnvWrapper`

DAgger VecEnvWrapper for querying and saving expert actions.

Every call to *.step(actions)* accepts and saves expert actions to *self.save_dir*, but only forwards expert actions to the wrapped VecEnv with probability *self.beta*. With probability *1 - self.beta*, a "robot" action (i.e an action from the imitation policy) is forwarded instead.

Demonstrations are saved as *TrajectoryWithRew* to *self.save_dir* at the end of every episode.

**__init__**(*venv*, *get_robot_acts*, *beta*, *save_dir*, *rng*)

Builds InteractiveTrajectoryCollector.

**Parameters**

- **venv** (`VecEnv`) – vectorized environment to sample trajectories from.
- **get_robot_acts** (`Callable[[ndarray], ndarray]`) – get robot actions that can be substituted for human actions. Takes a vector of observations as input & returns a vector of actions.
- **beta** (`float`) – fraction of the time to use action given to .step() instead of robot action. The choice of robot or human action is independently randomized for each individual *Env* at every timestep.
- **save_dir** (`Union[str, bytes, PathLike]`) – directory to save collected trajectories in.
- **rng** (`Generator`) – random state for random number generation.

**reset**()

Resets the environment.

**Returns**

first observation of a new trajectory.

**Return type**

obs

**seed**(*seed=None*)

Set the seed for the DAgger random number generator and wrapped VecEnv.

The DAgger RNG is used along with *self.beta* to determine whether the expert or robot action is forwarded to the wrapped VecEnv.

**Parameters**

**seed** (`Optional[int]`) – The random seed. May be None for completely random seeding.

**Return type**

`List[Optional[int]]`

**Returns**

A list containing the seeds for each individual env. Note that all list elements may be None, if the env does not return anything when seeded.

**step_async**(*actions*)

Steps with a *1 - beta* chance of using *self.get_robot_acts* instead.

DAgger needs to be able to inject imitation policy actions randomly at some subset of time steps. This method has a *self.beta* chance of keeping the *actions* passed in as an argument, and a *1 - self.beta* chance of forwarding actions generated by *self.get_robot_acts* instead. "robot" (i.e. imitation policy) action if necessary.

At the end of every episode, a *TrajectoryWithRew* is saved to *self.save_dir*, where every saved action is the expert action, regardless of whether the robot action was used during that timestep.

**Parameters**

**actions** (ndarray) – the _intended_ demonstrator/expert actions for the current state. This will be executed with probability *self.beta*. Otherwise, a "robot" (typically a BC policy) action will be sampled and executed instead via *self.get_robot_act*.

**Return type**

None

**step_wait**()

Returns observation, reward, etc after previous *step_async()* call.

Stores the transition, and saves trajectory as demo once complete.

**Return type**

Tuple[Union[ndarray, Dict[str, ndarray], Tuple[ndarray, ...]], ndarray, ndarray, List[Dict]]

**Returns**

Observation, reward, dones (is terminal?) and info dict.

**traj_accum: Optional[*TrajectoryAccumulator*]**

**class** imitation.algorithms.dagger.**LinearBetaSchedule**(*rampdown_rounds*)

Bases: *BetaSchedule*

Linearly-decreasing schedule for beta.

**__init__**(*rampdown_rounds*)

Builds LinearBetaSchedule.

**Parameters**

**rampdown_rounds** (int) – number of rounds over which to anneal beta.

**exception** imitation.algorithms.dagger.**NeedsDemosException**

Bases: Exception

Signals demos need to be collected for current round before continuing.

**class** imitation.algorithms.dagger.**SimpleDAggerTrainer**(**, venv, scratch_dir, expert_policy, rng, expert_trajs=None, **dagger_trainer_kwargs*)

Bases: *DAggerTrainer*

Simpler subclass of DAggerTrainer for training with synthetic feedback.

**__init__**(**, venv, scratch_dir, expert_policy, rng, expert_trajs=None, **dagger_trainer_kwargs*)

Builds SimpleDAggerTrainer.

**Parameters**

- **venv** (VecEnv) – Vectorized training environment. Note that when the robot action is randomly injected (in accordance with *beta_schedule* argument), every individual environment will get a robot action simultaneously for that timestep.

- **scratch_dir** (Union[str, bytes, PathLike]) – Directory to use to store intermediate training information (e.g. for resuming training).
- **expert_policy** (BasePolicy) – The expert policy used to generate synthetic demonstrations.
- **rng** (Generator) – Random state to use for the random number generator.
- **expert_trajs** (Optional[Sequence[*Trajectory*]]) – Optional starting dataset that is inserted into the round 0 dataset.
- **dagger_trainer_kwargs** – Other keyword arguments passed to the superclass initializer *DAggerTrainer.__init__*.

**Raises**

**ValueError** – The observation or action space does not match between *venv* and *expert_policy*.

**allow_variable_horizon: bool**

If True, allow variable horizon trajectories; otherwise error if detected.

**train** (*total_timesteps*, *, *rollout_round_min_episodes=3*, *rollout_round_min_timesteps=500*, *bc_train_kwargs=None*)

Train the DAgger agent.

The agent is trained in "rounds" where each round consists of a dataset aggregation step followed by BC update step.

During a dataset aggregation step, *self.expert_policy* is used to perform rollouts in the environment but there is a *1 - beta* chance (beta is determined from the round number and *self.beta_schedule*) that the DAgger agent's action is used instead. Regardless of whether the DAgger agent's action is used during the rollout, the expert action and corresponding observation are always appended to the dataset. The number of environment steps in the dataset aggregation stage is determined by the *rollout_round_min** arguments.

During a BC update step, *BC.train()* is called to update the DAgger agent on all data collected so far.

**Parameters**

- **total_timesteps** (int) – The number of timesteps to train inside the environment. In practice this is a lower bound, because the number of timesteps is rounded up to finish the minimum number of episodes or timesteps in the last DAgger training round, and the environment timesteps are executed in multiples of *self.venv.num_envs*.
- **rollout_round_min_episodes** (int) – The number of episodes the must be completed completed before a dataset aggregation step ends.
- **rollout_round_min_timesteps** (int) – The number of environment timesteps that must be completed before a dataset aggregation step ends. Also, that any round will always train for at least *self.batch_size* timesteps, because otherwise BC could fail to receive any batches.
- **bc_train_kwargs** (Optional[dict]) – Keyword arguments for calling *BC.train()*. If the *log_rollouts_venv* key is not provided, then it is set to *self.venv* by default. If neither of the *n_epochs* and *n_batches* keys are provided, then *n_epochs* is set to *self.DEFAULT_N_EPOCHS*.

**Return type**

None

imitation.algorithms.dagger.**reconstruct_trainer** (*scratch_dir*, *venv*, *custom_logger=None*, *device='auto'*)

Reconstruct trainer from the latest snapshot in some working directory.

Requires vectorized environment and (optionally) a logger, as these objects cannot be serialized.

**Parameters**

- **scratch_dir** (Union[str, bytes, PathLike]) – path to the working directory created by a previous run of this algorithm. The directory should contain *checkpoint-latest.pt* and *policy-latest.pt* files.
- **venv** (VecEnv) – Vectorized training environment.
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **device** (Union[device, str]) – device on which to load the trainer.

**Return type**

*DAggerTrainer*

**Returns**

A deserialized *DAggerTrainer*.

### imitation.algorithms.density

Density-based baselines for imitation learning.

Each of these algorithms learns a density estimate on some aspect of the demonstrations, then rewards the agent for following that estimate.

#### Classes

| | |
|---|---|
| *DensityAlgorithm*(*, demonstrations, venv, rng) | Learns a reward function based on density modeling. |
| *DensityType*(value) | Input type the density model should use. |

**class** imitation.algorithms.density.**DensityAlgorithm**(**, demonstrations, venv, rng, density_type=DensityType.STATE_ACTION_DENSITY, kernel='gaussian', kernel_bandwidth=0.5, rl_algo=None, is_stationary=True, standardise_inputs=True, custom_logger=None, allow_variable_horizon=False*)

Bases: *DemonstrationAlgorithm*

Learns a reward function based on density modeling.

Specifically, it constructs a non-parametric estimate of *p(s)*, *p(s,a)*, *p(s,s')* and then computes a reward using the log of these probabilities.

**__init__**(**, demonstrations, venv, rng, density_type=DensityType.STATE_ACTION_DENSITY, kernel='gaussian', kernel_bandwidth=0.5, rl_algo=None, is_stationary=True, standardise_inputs=True, custom_logger=None, allow_variable_horizon=False*)

Builds DensityAlgorithm.

**Parameters**

- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*, None]) – expert demonstration trajectories.
- **density_type** (*DensityType*) – type of density to train on: single state, state-action pairs, or state-state pairs.
- **kernel** (str) – kernel to use for density estimation with *sklearn.KernelDensity*.
- **kernel_bandwidth** (float) – bandwidth of kernel. If *standardise_inputs* is true and you are using a Gaussian kernel, then it probably makes sense to set this somewhere between 0.1 and 1.
- **venv** (VecEnv) – The environment to learn a reward model in. We don't actually need any environment interaction to fit the reward model, but we use this to extract the observation and action space, and to train the RL algorithm *rl_algo* (if specified).
- **rng** (Generator) – random state for sampling from demonstrations.
- **rl_algo** (Optional[BaseAlgorithm]) – An RL algorithm to train on the resulting reward model (optional).
- **is_stationary** (bool) – if True, share same density models for all timesteps; if False, use a different density model for each timestep. A non-stationary model is particularly likely to be useful when using STATE_DENSITY, to encourage agent to imitate entire trajectories, not just a few states that have high frequency in the demonstration dataset. If non-stationary, demonstrations must be trajectories, not transitions (which do not contain timesteps).
- **standardise_inputs** (bool) – if True, then the inputs to the reward model will be standardised to have zero mean and unit variance over the demonstration trajectories. Otherwise, inputs will be passed to the reward model with their ordinary scale.
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **allow_variable_horizon** (bool) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/guide/variable_horizon.html before overriding this.

**buffering_wrapper:** ***BufferingWrapper***

**density_type:** ***DensityType***

**is_stationary: bool**

**kernel: str**

**kernel_bandwidth: float**

**property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**

BasePolicy

**rl_algo: Optional[BaseAlgorithm]**

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

**Return type**

`None`

**standardise: bool**

**test_policy**(**, *n_trajectories=10*, *true_reward=True*)

Test current imitation policy on environment & give some rollout stats.

**Parameters**

- **n_trajectories** (`int`) – number of rolled-out trajectories.
- **true_reward** (`bool`) – should this use ground truth reward from underlying environment (True), or imitation reward (False)?

**Returns**

**rollout statistics collected by** *imitation.utils.rollout.rollout_stats()*.

**Return type**

dict

**train**()

Fits the density model to demonstration data *self.transitions*.

**Return type**

`None`

**train_policy**(*n_timesteps=1000000*, ***kwargs*)

Train the imitation policy for a given number of timesteps.

**Parameters**

- **n_timesteps** (`int`) – number of timesteps to train the policy for.
- **kwargs** (*dict*) – extra arguments that will be passed to the *learn()* method of the imitation RL model. Refer to Stable Baselines docs for details.

**Return type**

`None`

**transitions: Dict[Optional[int], ndarray]**

**venv: VecEnv**

**venv_wrapped:** ***RewardVecEnvWrapper***

**wrapper_callback:** ***WrappedRewardCallback***

**class** imitation.algorithms.density.**DensityType**(*value*)

Bases: `Enum`

Input type the density model should use.

**STATE_ACTION_DENSITY = 2**

Density on (s,a) pairs.

**STATE_DENSITY = 1**

Density on state s.

**STATE_STATE_DENSITY = 3**

Density on (s,s') pairs.

### imitation.algorithms.mce_irl

Finite-horizon tabular Maximum Causal Entropy IRL.

Follows the description in chapters 9 and 10 of Brian Ziebart's PhD thesis.

#### Functions

| | |
|---|---|
| *mce_occupancy_measures*(env, *[, reward, pi, ...]) | Calculate state visitation frequency Ds for each state s under a given policy pi. |
| *mce_partition_fh*(env, *[, reward, discount]) | Performs the soft Bellman backup for a finite-horizon MDP. |
| *squeeze_r*(r_output) | Squeeze a reward output tensor down to one dimension, if necessary. |

#### Classes

| | |
|---|---|
| *MCEIRL*(demonstrations, env, reward_net, rng) | Tabular MCE IRL. |
| *TabularPolicy*(state_space, action_space, pi, rng) | A tabular policy. |

**class** imitation.algorithms.mce_irl.**MCEIRL**(*demonstrations*, *env*, *reward_net*, *rng*, *optimizer_cls=<class 'torch.optim.adam.Adam'>*, *optimizer_kwargs=None*, *discount=1.0*, *linf_eps=0.001*, *grad_l2_eps=0.0001*, *log_interval=100*, ***, *custom_logger=None*)

Bases: *DemonstrationAlgorithm*[*TransitionsMinimal*]

Tabular MCE IRL.

Reward is a function of observations, but policy is a function of states.

The "observations" effectively exist just to let MCE IRL learn a reward in a reasonable feature space, giving a helpful inductive bias, e.g. that similar states have similar reward.

Since we are performing planning to compute the policy, there is no need for function approximation in the policy.

**__init__**(*demonstrations*, *env*, *reward_net*, *rng*, *optimizer_cls=<class 'torch.optim.adam.Adam'>*, *optimizer_kwargs=None*, *discount=1.0*, *linf_eps=0.001*, *grad_l2_eps=0.0001*, *log_interval=100*, ***, *custom_logger=None*)

Creates MCE IRL.

**Parameters**

- **demonstrations** (Union[ndarray, Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*, None]) – Demonstrations from an expert (optional). Can be a sequence of trajectories, or transitions, an iterable over mappings that represent a batch of transitions, or a state occupancy measure. The demonstrations must have observations one-hot coded unless demonstrations is a state-occupancy measure.

- **env** (`TabularModelPOMDP`) – a tabular MDP.
- **rng** (`Generator`) – random state used for sampling from policy.
- **reward_net** (*`RewardNet`*) – a neural network that computes rewards for the supplied observations.
- **optimizer_cls** (`Type[Optimizer]`) – optimizer to use for supervised training.
- **optimizer_kwargs** (`Optional[Mapping[str, Any]]`) – keyword arguments for optimizer construction.
- **discount** (`float`) – the discount factor to use when computing occupancy measure. If not 1.0 (undiscounted), then *demonstrations* must either be a (discounted) state-occupancy measure, or trajectories. Transitions are *not allowed* as we cannot discount them appropriately without knowing the timestep they were drawn from.
- **linf_eps** (`float`) – optimisation terminates if the $l_{infty}$ distance between the demonstrator's state occupancy measure and the state occupancy measure for the current reward falls below this value.
- **grad_l2_eps** (`float`) – optimisation also terminates if the $ell_2$ norm of the MCE IRL gradient falls below this value.
- **log_interval** (`Optional[int]`) – how often to log current loss stats (using *logging*). None to disable.
- **custom_logger** (`Optional[`*`HierarchicalLogger`*`]`) – Where to log to; if None (default), creates a new logger.

**Raises**

**ValueError** – if the env horizon is not finite (or an integer).

**demo_state_om: Optional[ndarray]**

**property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**

`BasePolicy`

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (`Union[ndarray, Iterable[`*`Trajectory`*`], Iterable[`*`TransitionMapping`*`], `*`TransitionsMinimal`*`]`) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing "obs" and "acts" Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**

`None`

**train**(*max_iter=1000*)

Runs MCE IRL.

**Parameters**

**max_iter** (`int`) – The maximum number of iterations to train for. May terminate earlier if *self.linf_eps* or *self.grad_l2_eps* thresholds are reached.

**Return type**
`ndarray`

**Returns**
State occupancy measure for the final reward function. *self.reward_net* and *self.optimizer* will be updated in-place during optimisation.

**class** `imitation.algorithms.mce_irl.`**`TabularPolicy`**(*state_space*, *action_space*, *pi*, *rng*)

Bases: `BasePolicy`

A tabular policy. Cannot be trained – prediction only.

**`__init__`**(*state_space*, *action_space*, *pi*, *rng*)

Builds TabularPolicy.

**Parameters**

- **`state_space`** (`Space`) – The state space of the environment.
- **`action_space`** (`Space`) – The action space of the environment.
- **`pi`** (`ndarray`) – A tabular policy. Three-dimensional array, where pi[t,s,a] is the probability of taking action a at state s at timestep t.
- **`rng`** (`Generator`) – Random state, used for sampling when *predict* is called with *deterministic=False*.

**`forward`**(*observation*, *deterministic=False*)

Define 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.

**Return type**
`NoReturn`

**`pi: ndarray`**

**`predict`**(*observation*, *state=None*, *episode_start=None*, *deterministic=False*)

Predict action to take in given state.

Arguments follow SB3 naming convention as this is an SB3 policy. In this convention, observations are returned by the environment, and state is a hidden state used by the policy (used by us to keep track of timesteps).

What is *observation* here is a state in the underlying MDP, and would be called *state* elsewhere in this file.

**Parameters**

- **`observation`** (`Union[ndarray, Mapping[str, ndarray]]`) – States in the underlying MDP.
- **`state`** (`Optional[Tuple[ndarray, ...]]`) – Hidden states of the policy – used to represent timesteps by us.
- **`episode_start`** (`Optional[ndarray]`) – Has episode completed?

- **deterministic** (bool) – If true, pick action with highest probability; otherwise, sample.

**Return type**
Tuple[ndarray, Optional[Tuple[ndarray, ...]]]

**Returns**
Tuple of the actions and new hidden states.

**rng: Generator**

**set_pi**(*pi*)

Sets tabular policy to *pi*.

**Return type**
None

imitation.algorithms.mce_irl.**mce_occupancy_measures**(*env*, *, *reward=None*, *pi=None*, *discount=1.0*)

Calculate state visitation frequency Ds for each state s under a given policy pi.

You can get pi from *mce_partition_fh*.

**Parameters**

- **env** (TabularModelPOMDP) – a tabular MDP.
- **reward** (Optional[ndarray]) – reward matrix. Defaults is env.reward_matrix.
- **pi** (Optional[ndarray]) – policy to simulate. Defaults to soft-optimal policy w.r.t reward matrix.
- **discount** (float) – rate to discount the cumulative occupancy measure D.

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

**Returns**
Tuple of D (ndarray) and Dcum (ndarray). D is of shape (env.horizon, env.n_states) and records the probability of being in a given state at a given timestep. Dcum is of shape (env.n_states,) and records the expected discounted number of times each state is visited.

**Raises**
**ValueError** – if env.horizon is None (infinite horizon).

imitation.algorithms.mce_irl.**mce_partition_fh**(*env*, *, *reward=None*, *discount=1.0*)

Performs the soft Bellman backup for a finite-horizon MDP.

Calculates V^{soft}, Q^{soft}, and pi using recurrences (9.1), (9.2), and (9.3) from Ziebart (2010).

**Parameters**

- **env** (TabularModelPOMDP) – a tabular, known-dynamics MDP.
- **reward** (Optional[ndarray]) – a reward matrix. Defaults to env.reward_matrix.
- **discount** (float) – discount rate.

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

**Returns**
(V, Q, pi) corresponding to the soft values, Q-values and MCE policy. V is a 2d array, indexed V[t,s]. Q is a 3d array, indexed Q[t,s,a]. pi is a 3d array, indexed pi[t,s,a].

**Raises**

**`ValueError`** – if `env.horizon` is None (infinite horizon).

`imitation.algorithms.mce_irl.`**`squeeze_r`**(*r_output*)

Squeeze a reward output tensor down to one dimension, if necessary.

**Parameters**

**`r_output`** (*`th.Tensor`*) – output of reward model. Can be either 1D ([n_states]) or 2D ([n_states, 1]).

**Return type**

`Tensor`

**Returns**

squeezed reward of shape [n_states].

### imitation.algorithms.preference_comparisons

Learning reward models using preference comparisons.

Trains a reward model and optionally a policy based on preferences between trajectory fragments.

#### Functions

| | |
|---|---|
| *`get_base_model`*(reward_model) | **rtype**<br>*`RewardNet`* |
| *`preference_collate_fn`*(batch) | **rtype**<br>`Tuple[Sequence[Tuple[`*`TrajectoryWithRew`*`, `*`TrajectoryWithRew`*`]], ndarray]` |

### Classes

| | |
|---|---|
| `ActiveSelectionFragmenter`(preference_model, ...) | Sample fragments of trajectories based on active selection. |
| `AgentTrainer`(algorithm, reward_fn, venv, rng) | Wrapper for training an SB3 algorithm on an arbitrary reward function. |
| `BasicRewardTrainer`(preference_model, loss, rng) | Train a basic reward model. |
| `CrossEntropyRewardLoss`() | Compute the cross entropy reward loss. |
| `EnsembleTrainer`(preference_model, loss, rng) | Train a reward ensemble. |
| `Fragmenter`([custom_logger]) | Class for creating pairs of trajectory fragments from a set of trajectories. |
| `LossAndMetrics`(loss, metrics) | Loss and auxiliary metrics for reward network training. |
| `PreferenceComparisons`(trajectory_generator, ...) | Main interface for reward learning using preference comparisons. |
| `PreferenceDataset`([max_size]) | A PyTorch Dataset for preference comparisons. |
| `PreferenceGatherer`([rng, custom_logger]) | Base class for gathering preference comparisons between trajectory fragments. |
| `PreferenceModel`(model[, noise_prob, ...]) | Class to convert two fragments' rewards into preference probability. |
| `RandomFragmenter`(rng[, warning_threshold, ...]) | Sample fragments of trajectories uniformly at random with replacement. |
| `RewardLoss`(*args, **kwargs) | A loss function over preferences. |
| `RewardTrainer`(preference_model[, custom_logger]) | Abstract base class for training reward models using preference comparisons. |
| `SyntheticGatherer`([temperature, ...]) | Computes synthetic preferences using ground-truth environment rewards. |
| `TrajectoryDataset`(trajectories, rng[, ...]) | A fixed dataset of trajectories. |
| `TrajectoryGenerator`([custom_logger]) | Generator of trajectories with optional training logic. |

**class** imitation.algorithms.preference_comparisons.**ActiveSelectionFragmenter**(*preference_model*, *base_fragmenter*, *fragment_sample_factor*, *uncertainty_on='logit'*, *custom_logger=None*)

Bases: `Fragmenter`

Sample fragments of trajectories based on active selection.

Actively picks the fragment pairs with the highest uncertainty (variance) of rewards/probabilties/predictions from ensemble model.

**__init__**(*preference_model*, *base_fragmenter*, *fragment_sample_factor*, *uncertainty_on='logit'*, *custom_logger=None*)

Initialize the active selection fragmenter.

**Parameters**

- **preference_model** (*PreferenceModel*) – an ensemble model that predicts the preference of the first fragment over the other.
- **base_fragmenter** (*Fragmenter*) – fragmenter instance to get fragment pairs from trajectories
- **fragment_sample_factor** (`float`) – the factor of the number of fragment pairs to sample from the base_fragmenter
- **uncertainty_on** (`str`) – the variable to calculate the variance on. Can be logit|probability|label.
- **custom_logger** (`Optional[`*HierarchicalLogger*`]`) – Where to log to; if None (default), creates a new logger.

**Raises**

**ValueError** – Preference model not wrapped over an ensemble of networks.

**raise_uncertainty_on_not_supported**()

**Return type**

`NoReturn`

**property uncertainty_on: str**

**Return type**

`str`

**variance_estimate**(*rews1*, *rews2*)

Gets the variance estimate from the rewards of a fragment pair.

**Parameters**

- **rews1** (`Tensor`) – rewards obtained by all the ensemble models for the first fragment. Shape - (fragment_length, num_ensemble_members)
- **rews2** (`Tensor`) – rewards obtained by all the ensemble models for the second fragment. Shape - (fragment_length, num_ensemble_members)

**Return type**

`float`

**Returns**

the variance estimate based on the *uncertainty_on* flag.

**class** imitation.algorithms.preference_comparisons.**AgentTrainer**(*algorithm*, *reward_fn*, *venv*, *rng*, *exploration_frac=0.0*, *switch_prob=0.5*, *random_prob=0.5*, *custom_logger=None*)

Bases: *TrajectoryGenerator*

Wrapper for training an SB3 algorithm on an arbitrary reward function.

**__init__**(*algorithm*, *reward_fn*, *venv*, *rng*, *exploration_frac=0.0*, *switch_prob=0.5*, *random_prob=0.5*, *custom_logger=None*)

Initialize the agent trainer.

**Parameters**

- **algorithm** (BaseAlgorithm) – the stable-baselines algorithm to use for training.
- **reward_fn** (Union[*RewardFn*, *RewardNet*]) – either a RewardFn or a RewardNet instance that will supply the rewards used for training the agent.
- **venv** (VecEnv) – vectorized environment to train in.
- **rng** (Generator) – random number generator used for exploration and for sampling.
- **exploration_frac** (float) – fraction of the trajectories that will be generated partially randomly rather than only by the agent when sampling.
- **switch_prob** (float) – the probability of switching the current policy at each step for the exploratory samples.
- **random_prob** (float) – the probability of picking the random policy when switching during exploration.
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.

**property logger:** ***HierarchicalLogger***

**Return type**

*HierarchicalLogger*

**sample**(*steps*)

Sample a batch of trajectories.

**Parameters**

**steps** (int) – All trajectories taken together should have at least this many steps.

**Return type**

Sequence[*TrajectoryWithRew*]

**Returns**

A list of sampled trajectories with rewards (which should be the environment rewards, not ones from a reward model).

**train**(*steps*, ***kwargs*)

Train the agent using the reward function specified during instantiation.

**Parameters**

- **steps** (int) – number of environment timesteps to train for
- ****kwargs** – other keyword arguments to pass to BaseAlgorithm.train()

**Raises**

**RuntimeError** – Transitions left in *self.buffering_wrapper*; call *self.sample* first to clear them.

**Return type**

None

**class** imitation.algorithms.preference_comparisons.**BasicRewardTrainer**(*preference_model*, *loss*, *rng*, *batch_size=32*, *minibatch_size=None*, *epochs=1*, *lr=0.001*, *custom_logger=None*, *regularizer_factory=None*)

Bases: *RewardTrainer*

Train a basic reward model.

**__init__**(*preference_model*, *loss*, *rng*, *batch_size=32*, *minibatch_size=None*, *epochs=1*, *lr=0.001*, *custom_logger=None*, *regularizer_factory=None*)

Initialize the reward model trainer.

**Parameters**

- **preference_model** (*PreferenceModel*) – the preference model to train the reward network.
- **loss** (*RewardLoss*) – the loss to use
- **rng** (Generator) – the random number generator to use for splitting the dataset into training and validation.
- **batch_size** (int) – number of fragment pairs per batch
- **minibatch_size** (Optional[int]) – size of minibatch to calculate gradients over. The gradients are accumulated until *batch_size* examples are processed before making an optimization step. This is useful in GPU training to reduce memory usage, since fewer examples are loaded into memory at once, facilitating training with larger batch sizes, but is generally slower. Must be a factor of *batch_size*. Optional, defaults to *batch_size*.
- **epochs** (int) – number of epochs in each training iteration (can be adjusted on the fly by specifying an *epoch_multiplier* in *self.train()* if longer training is desired in specific cases).
- **lr** (float) – the learning rate
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **regularizer_factory** (Optional[*RegularizerFactory*]) – if you would like to apply regularization during training, specify a regularizer factory here. The factory will be used to construct a regularizer. See imitation.regularization.RegularizerFactory for more details.

**Raises**

**ValueError** – if the batch size is not a multiple of the minibatch size.

**regularizer: Optional[*Regularizer*]**

**property requires_regularizer_update: bool**

Whether the regularizer requires updating.

**Return type**
`bool`

**Returns**
If true, this means that a validation dataset will be used.

**class** `imitation.algorithms.preference_comparisons.`**`CrossEntropyRewardLoss`**

Bases: *`RewardLoss`*

Compute the cross entropy reward loss.

**`__init__`**`()`

Create cross entropy reward loss.

**`forward`**(*fragment_pairs*, *preferences*, *preference_model*)

Computes the loss.

**Parameters**

- **`fragment_pairs`** (`Sequence[Tuple[`*`Trajectory`*`, `*`Trajectory`*`]]`) – Batch consisting of pairs of trajectory fragments.
- **`preferences`** (`ndarray`) – The probability that the first fragment is preferred over the second. Typically 0, 1 or 0.5 (tie).
- **`preference_model`** (*`PreferenceModel`*) – model to predict the preferred fragment from a pair.

**Return type**
*`LossAndMetrics`*

**Returns**

**The cross-entropy loss between the probability predicted by the**
reward model and the target probabilities in *preferences*. Metrics are accuracy, and gt_reward_loss, if the ground truth reward is available.

**`training: bool`**

**class** `imitation.algorithms.preference_comparisons.`**`EnsembleTrainer`**(*preference_model*, *loss*, *rng*, *batch_size=32*, *minibatch_size=None*, *epochs=1*, *lr=0.001*, *custom_logger=None*, *regularizer_factory=None*)

Bases: *`BasicRewardTrainer`*

Train a reward ensemble.

**`__init__`**(*preference_model*, *loss*, *rng*, *batch_size=32*, *minibatch_size=None*, *epochs=1*, *lr=0.001*, *custom_logger=None*, *regularizer_factory=None*)

Initialize the reward model trainer.

**Parameters**

- **`preference_model`** (*`PreferenceModel`*) – the preference model to train the reward network.

- **loss** (*RewardLoss*) – the loss to use
- **rng** (Generator) – random state for the internal RNG used in bagging
- **batch_size** (int) – number of fragment pairs per batch
- **minibatch_size** (Optional[int]) – size of minibatch to calculate gradients over. The gradients are accumulated until *batch_size* examples are processed before making an optimization step. This is useful in GPU training to reduce memory usage, since fewer examples are loaded into memory at once, facilitating training with larger batch sizes, but is generally slower. Must be a factor of *batch_size*. Optional, defaults to *batch_size*.
- **epochs** (int) – number of epochs in each training iteration (can be adjusted on the fly by specifying an *epoch_multiplier* in *self.train()* if longer training is desired in specific cases).
- **lr** (float) – the learning rate
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **regularizer_factory** (Optional[*RegularizerFactory*]) – A factory for creating a regularizer. If None, no regularization is used.

**Raises**

**TypeError** – if model is not a RewardEnsemble.

**property logger:** ***HierarchicalLogger***

**Return type**

*HierarchicalLogger*

**regularizer: Optional[*Regularizer*]**

**class** imitation.algorithms.preference_comparisons.**Fragmenter**(*custom_logger=None*)

Bases: ABC

Class for creating pairs of trajectory fragments from a set of trajectories.

**__init__**(*custom_logger=None*)

Initialize the fragmenter.

**Parameters**

**custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.

**class** imitation.algorithms.preference_comparisons.**LossAndMetrics**(*loss: Tensor, metrics: Mapping[str, Tensor]*)

Bases: tuple

Loss and auxiliary metrics for reward network training.

**loss: Tensor**

**metrics: Mapping[str, Tensor]**

**class** imitation.algorithms.preference_comparisons.**PreferenceComparisons**(*trajectory_generator*, *reward_model*, *num_iterations*, *fragmenter=None*, *preference_gatherer=None*, *reward_trainer=None*, *comparison_queue_size=None*, *fragment_length=100*, *transition_oversampling=1*, *initial_comparison_frac=0.1*, *initial_epoch_multiplier=200.0*, *custom_logger=None*, *allow_variable_horizon=False*, *rng=None*, *query_schedule='hyperbolic'*)

Bases: `BaseImitationAlgorithm`

Main interface for reward learning using preference comparisons.

**__init__**(*trajectory_generator*, *reward_model*, *num_iterations*, *fragmenter=None*, *preference_gatherer=None*, *reward_trainer=None*, *comparison_queue_size=None*, *fragment_length=100*, *transition_oversampling=1*, *initial_comparison_frac=0.1*, *initial_epoch_multiplier=200.0*, *custom_logger=None*, *allow_variable_horizon=False*, *rng=None*, *query_schedule='hyperbolic'*)

Initialize the preference comparison trainer.

The loggers of all subcomponents are overridden with the logger used by this class.

**Parameters**

- **trajectory_generator** (*TrajectoryGenerator*) – generates trajectories while optionally training an RL agent on the learned reward function (can also be a sampler from a static dataset of trajectories though).
- **reward_model** (*RewardNet*) – a RewardNet instance to be used for learning the reward
- **num_iterations** (int) – number of times to train the agent against the reward model and then train the reward model against newly gathered preferences.
- **fragmenter** (Optional[*Fragmenter*]) – takes in a set of trajectories and returns pairs of fragments for which preferences will be gathered. These fragments could be random, or they could be selected more deliberately (active learning). Default is a random fragmenter.
- **preference_gatherer** (Optional[*PreferenceGatherer*]) – how to get preferences between trajectory fragments. Default (and currently the only option) is to use synthetic preferences based on ground-truth rewards. Human preferences could be implemented here in the future.
- **reward_trainer** (Optional[*RewardTrainer*]) – trains the reward model based on pairs of fragments and associated preferences. Default is to use the preference model and loss function from DRLHP.
- **comparison_queue_size** (Optional[int]) – the maximum number of comparisons to keep in the queue for training the reward model. If None, the queue will grow without bound as new comparisons are added.
- **fragment_length** (int) – number of timesteps per fragment that is used to elicit preferences
- **transition_oversampling** (float) – factor by which to oversample transitions before creating fragments. Since fragments are sampled with replacement, this is usually chosen > 1 to avoid having the same transition in too many fragments.
- **initial_comparison_frac** (float) – fraction of the total_comparisons argument to train() that will be sampled before the rest of training begins (using a randomly initialized agent). This can be used to pretrain the reward model before the agent is trained on the learned reward, to help avoid irreversibly learning a bad policy from an untrained reward. Note that there will often be some additional pretraining comparisons since *comparisons_per_iteration* won't exactly divide the total number of comparisons. How many such comparisons there are depends discontinuously on *total_comparisons* and *comparisons_per_iteration*.
- **initial_epoch_multiplier** (float) – before agent training begins, train the reward model for this many more epochs than usual (on fragments sampled from a random agent).
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **allow_variable_horizon** (bool) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/guide/variable_horizon.html before overriding this.
- **rng** (Optional[Generator]) – random number generator to use for initializing subcomponents such as fragmenter. Only used when default components are used; if you instantiate your own fragmenter, preference gatherer, etc., you are responsible for seeding them!
- **query_schedule** (Union[str, Callable[[float], float]]) – one of ("constant", "hyperbolic", "inverse_quadratic"), or a function that takes in a float between 0 and 1 inclu-

sive, representing a fraction of the total number of timesteps elapsed up to some time T, and returns a potentially unnormalized probability indicating the fraction of *total_comparisons* that should be queried at that iteration. This function will be called *num_iterations* times in *__init__()* with values from *np.linspace(0, 1, num_iterations)* as input. The outputs will be normalized to sum to 1 and then used to apportion the comparisons among the *num_iterations* iterations.

**Raises**

`ValueError` – if *query_schedule* is not a valid string or callable.

**`allow_variable_horizon: bool`**

If True, allow variable horizon trajectories; otherwise error if detected.

**`train`**(*total_timesteps*, *total_comparisons*, *callback=None*)

Train the reward model and the policy if applicable.

**Parameters**

- **`total_timesteps`** (`int`) – number of environment interaction steps
- **`total_comparisons`** (`int`) – number of preferences to gather in total
- **`callback`** (`Optional[Callable[[int], None]]`) – callback functions called at the end of each iteration

**Return type**

`Mapping[str, Any]`

**Returns**

A dictionary with final metrics such as loss and accuracy of the reward model.

**class** `imitation.algorithms.preference_comparisons.`**`PreferenceDataset`**(*max_size=None*)

Bases: `Dataset`

A PyTorch Dataset for preference comparisons.

Each item is a tuple consisting of two trajectory fragments and a probability that fragment 1 is preferred over fragment 2.

This dataset is meant to be generated piece by piece during the training process, which is why data can be added via the .push() method.

**`__init__`**(*max_size=None*)

Builds an empty PreferenceDataset.

**Parameters**

**`max_size`** (`Optional[int]`) – Maximum number of preference comparisons to store in the dataset. If None (default), the dataset can grow indefinitely. Otherwise, the dataset acts as a FIFO queue, and the oldest comparisons are evicted when *push()* is called and the dataset is at max capacity.

**`static load`**(*path*)

**Return type**

*`PreferenceDataset`*

**`push`**(*fragments*, *preferences*)

Add more samples to the dataset.

**Parameters**

- **`fragments`** (`Sequence[Tuple[`*`TrajectoryWithRew`*`, `*`TrajectoryWithRew`*`]]`) – list of pairs of trajectory fragments to add

- **preferences** (ndarray) – corresponding preference probabilities (probability that fragment 1 is preferred over fragment 2)

**Raises**
`ValueError` – *preferences* shape does not match *fragments* or has non-float32 dtype.

**Return type**
None

**save** (*path*)

**Return type**
None

**class** imitation.algorithms.preference_comparisons.**PreferenceGatherer** (*rng=None*, *custom_logger=None*)

Bases: ABC

Base class for gathering preference comparisons between trajectory fragments.

**__init__** (*rng=None*, *custom_logger=None*)

Initializes the preference gatherer.

**Parameters**

- **rng** (Optional[Generator]) – random number generator, if applicable.
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.

**class** imitation.algorithms.preference_comparisons.**PreferenceModel** (*model*, *noise_prob=0.0*, *discount_factor=1.0*, *threshold=50*)

Bases: Module

Class to convert two fragments' rewards into preference probability.

**__init__** (*model*, *noise_prob=0.0*, *discount_factor=1.0*, *threshold=50*)

Create Preference Prediction Model.

**Parameters**

- **model** (*RewardNet*) – base model to compute reward.
- **noise_prob** (float) – assumed probability with which the preference is uniformly random (used for the model of preference generation that is used for the loss).
- **discount_factor** (float) – the model of preference generation uses a softmax of returns as the probability that a fragment is preferred. This is the discount factor used to calculate those returns. Default is 1, i.e. undiscounted sums of rewards (which is what the DRLHP paper uses).
- **threshold** (float) – the preference model used to compute the loss contains a softmax of returns. To avoid overflows, we clip differences in returns that are above this threshold. This threshold is therefore in logspace. The default value of 50 means that probabilities below 2e-22 are rounded up to 2e-22.

**Raises**

`ValueError` – if *RewardEnsemble* is wrapped around a class other than *AddSTDRewardWrapper*.

`forward`(*fragment_pairs*)

Computes the preference probability of the first fragment for all pairs.

**Note: This function passes the gradient through for non-ensemble models.**

For an ensemble model, this function should not be used for loss calculation. It can be used in case where passing the gradient is not required such as during active selection or inference time. Therefore, the EnsembleTrainer passes each member network through this function instead of passing the EnsembleNetwork object with the use of *ensemble_member_index*.

**Parameters**

`fragment_pairs` (`Sequence[Tuple[`*`Trajectory`*`,` *`Trajectory`*`]]`) – batch of pair of fragments.

**Return type**

`Tuple[Tensor, Optional[Tensor]]`

**Returns**

A tuple with the first element as the preference probabilities for the first fragment for all fragment pairs given by the network(s). If the ground truth rewards are available, it also returns gt preference probabilities in the second element of the tuple (else None). Reward probability shape - (num_fragment_pairs, ) for non-ensemble reward network and (num_fragment_pairs, num_networks) for an ensemble of networks.

`probability`(*rews1*, *rews2*)

Computes the Boltzmann rational probability the first trajectory is best.

**Parameters**

- **`rews1`** (`Tensor`) – array/matrix of rewards for the first trajectory fragment. matrix for ensemble models and array for non-ensemble models.
- **`rews2`** (`Tensor`) – array/matrix of rewards for the second trajectory fragment. matrix for ensemble models and array for non-ensemble models.

**Return type**

`Tensor`

**Returns**

The softmax of the difference between the (discounted) return of the first and second trajectory. Shape - (num_ensemble_members, ) for ensemble model and () for non-ensemble model which is a torch scalar.

`rewards`(*transitions*)

Computes the reward for all transitions.

**Parameters**

`transitions` (*`Transitions`*) – batch of obs-act-obs-done for a fragment of a trajectory.

**Return type**

`Tensor`

**Returns**

The reward given by the network(s) for all the transitions. Shape - (num_transitions, ) for Single reward network and (num_transitions, num_networks) for ensemble of networks.

**training: bool**

**class** imitation.algorithms.preference_comparisons.**RandomFragmenter**(*rng*, *warning_threshold=10*, *custom_logger=None*)

Bases: `Fragmenter`

Sample fragments of trajectories uniformly at random with replacement.

Note that each fragment is part of a single episode and has a fixed length. This leads to a bias: transitions at the beginning and at the end of episodes are less likely to occur as part of fragments (this affects the first and last fragment_length transitions).

An additional bias is that trajectories shorter than the desired fragment length are never used.

**__init__**(*rng*, *warning_threshold=10*, *custom_logger=None*)

Initialize the fragmenter.

**Parameters**

- **rng** (`Generator`) – the random number generator
- **warning_threshold** (`int`) – give a warning if the number of available transitions is less than this many times the number of required samples. Set to 0 to disable this warning.
- **custom_logger** (`Optional[HierarchicalLogger]`) – Where to log to; if None (default), creates a new logger.

**class** imitation.algorithms.preference_comparisons.**RewardLoss**(**args*, ***kwargs*)

Bases: `Module`, `ABC`

A loss function over preferences.

**abstract forward**(*fragment_pairs*, *preferences*, *preference_model*)

Computes the loss.

**Parameters**

- **fragment_pairs** (`Sequence[Tuple[Trajectory, Trajectory]]`) – Batch consisting of pairs of trajectory fragments.
- **preferences** (`ndarray`) – The probability that the first fragment is preferred over the second. Typically 0, 1 or 0.5 (tie).
- **preference_model** (`PreferenceModel`) – model to predict the preferred fragment from a pair.

**Returns: # noqa: DAR202**

loss: the loss metrics: a dictionary of metrics that can be logged

**Return type**

`LossAndMetrics`

**training: bool**

**class** imitation.algorithms.preference_comparisons.**RewardTrainer**(*preference_model*, *custom_logger=None*)

Bases: `ABC`

Abstract base class for training reward models using preference comparisons.

This class contains only the actual reward model training code, it is not responsible for gathering trajectories and preferences or for agent training (see :class: *PreferenceComparisons* for that).

**`__init__`**(*preference_model*, *custom_logger=None*)

Initialize the reward trainer.

**Parameters**

- **`preference_model`** (`PreferenceModel`) – the preference model to train the reward network.
- **`custom_logger`** (`Optional[HierarchicalLogger]`) – Where to log to; if None (default), creates a new logger.

**`property logger: HierarchicalLogger`**

**Return type**

`HierarchicalLogger`

**`train`**(*dataset*, *epoch_multiplier=1.0*)

Train the reward model on a batch of fragment pairs and preferences.

**Parameters**

- **`dataset`** (`PreferenceDataset`) – the dataset of preference comparisons to train on.
- **`epoch_multiplier`** (`float`) – how much longer to train for than usual (measured relatively).

**Return type**

`None`

**`class`** `imitation.algorithms.preference_comparisons.`**`SyntheticGatherer`**(*temperature=1*, *discount_factor=1*, *sample=True*, *rng=None*, *threshold=50*, *custom_logger=None*)

Bases: `PreferenceGatherer`

Computes synthetic preferences using ground-truth environment rewards.

**`__init__`**(*temperature=1*, *discount_factor=1*, *sample=True*, *rng=None*, *threshold=50*, *custom_logger=None*)

Initialize the synthetic preference gatherer.

**Parameters**

- **`temperature`** (`float`) – the preferences are sampled from a softmax, this is the temperature used for sampling. temperature=0 leads to deterministic results (for equal rewards, 0.5 will be returned).
- **`discount_factor`** (`float`) – discount factor that is used to compute how good a fragment is. Default is to use undiscounted sums of rewards (as in the DRLHP paper).

- **sample** (bool) – if True (default), the preferences are 0 or 1, sampled from a Bernoulli distribution (or 0.5 in the case of ties with zero temperature). If False, then the underlying Bernoulli probabilities are returned instead.
- **rng** (Optional[Generator]) – random number generator, only used if `temperature > 0` and `sample=True`
- **threshold** (float) – preferences are sampled from a softmax of returns. To avoid overflows, we clip differences in returns that are above this threshold (after multiplying with temperature). This threshold is therefore in logspace. The default value of 50 means that probabilities below 2e-22 are rounded up to 2e-22.
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.

**Raises**

**ValueError** – if *sample* is true and no random state is provided.

**class** imitation.algorithms.preference_comparisons.**TrajectoryDataset**(*trajectories*, *rng*, *custom_logger=None*)

Bases: *TrajectoryGenerator*

A fixed dataset of trajectories.

**__init__**(*trajectories*, *rng*, *custom_logger=None*)

Creates a dataset loaded from *path*.

**Parameters**

- **trajectories** (Sequence[*TrajectoryWithRew*]) – the dataset of rollouts.
- **rng** (Generator) – RNG used for shuffling dataset.
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.

**sample**(*steps*)

Sample a batch of trajectories.

**Parameters**

**steps** (int) – All trajectories taken together should have at least this many steps.

**Return type**

Sequence[*TrajectoryWithRew*]

**Returns**

A list of sampled trajectories with rewards (which should be the environment rewards, not ones from a reward model).

**class** imitation.algorithms.preference_comparisons.**TrajectoryGenerator**(*custom_logger=None*)

Bases: ABC

Generator of trajectories with optional training logic.

**__init__**(*custom_logger=None*)

Builds TrajectoryGenerator.

**Parameters**

`custom_logger` (`Optional[HierarchicalLogger]`) – Where to log to; if None (default), creates a new logger.

`property logger: HierarchicalLogger`

**Return type**

`HierarchicalLogger`

`abstract sample(steps)`

Sample a batch of trajectories.

**Parameters**

`steps` (`int`) – All trajectories taken together should have at least this many steps.

**Return type**

`Sequence[TrajectoryWithRew]`

**Returns**

A list of sampled trajectories with rewards (which should be the environment rewards, not ones from a reward model).

`train(steps, **kwargs)`

Train an agent if the trajectory generator uses one.

By default, this method does nothing and doesn't need to be overridden in subclasses that don't require training.

**Parameters**

- `steps` (`int`) – number of environment steps to train for.
- `**kwargs` – additional keyword arguments to pass on to the training procedure.

**Return type**

`None`

`imitation.algorithms.preference_comparisons.get_base_model(reward_model)`

**Return type**

`RewardNet`

`imitation.algorithms.preference_comparisons.preference_collate_fn(batch)`

**Return type**

`Tuple[Sequence[Tuple[TrajectoryWithRew, TrajectoryWithRew]], ndarray]`

### imitation.algorithms.sqil

Soft Q Imitation Learning (SQIL) (https://arxiv.org/abs/1905.11108).

Trains a policy via DQN-style Q-learning, replacing half the buffer with expert demonstrations and adjusting the rewards.

## Classes

| | |
|---|---|
| *SQIL*(*, venv, demonstrations, policy[, ...]) | Soft Q Imitation Learning (SQIL). |
| *SQILReplayBuffer*(buffer_size, ...[, device, ...]) | A replay buffer that injects 50% expert demonstrations when sampling. |

**class** imitation.algorithms.sqil.**SQIL**(**, venv, demonstrations, policy, custom_logger=None, rl_algo_class=<class 'stable_baselines3.dqn.dqn.DQN'>, rl_kwargs=None*)

Bases: *DemonstrationAlgorithm*[*Transitions*]

Soft Q Imitation Learning (SQIL).

Trains a policy via DQN-style Q-learning, replacing half the buffer with expert demonstrations and adjusting the rewards.

**__init__**(**, venv, demonstrations, policy, custom_logger=None, rl_algo_class=<class 'stable_baselines3.dqn.dqn.DQN'>, rl_kwargs=None*)

Builds SQIL.

**Parameters**

- **venv** (VecEnv) – The vectorized environment to train on.
- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*, None]) – Demonstrations to use for training.
- **policy** (Union[str, Type[BasePolicy]]) – The policy model to use (SB3).
- **custom_logger** (Optional[*HierarchicalLogger*]) – Where to log to; if None (default), creates a new logger.
- **rl_algo_class** (Type[OffPolicyAlgorithm]) – Off-policy RL algorithm to use.
- **rl_kwargs** (Optional[Dict[str, Any]]) – Keyword arguments to pass to the RL algorithm constructor.

**Raises**

**ValueError** – if *dqn_kwargs* includes a key *replay_buffer_class* or *replay_buffer_kwargs*.

**expert_buffer: ReplayBuffer**

**property policy: BasePolicy**

Returns a policy imitating the demonstration data.

**Return type**

BasePolicy

**set_demonstrations**(*demonstrations*)

Sets the demonstration data.

Changing the demonstration data on-demand can be useful for interactive algorithms like DAgger.

**Parameters**

**demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Either a Torch *DataLoader*, any other iterator that yields dictionaries containing “obs” and “acts” Tensors or NumPy arrays, *TransitionKind* instance, or a Sequence of Trajectory objects.

**Return type**
None

**train**(*, *total_timesteps*, *tb_log_name='SQIL'*, ***kwargs*)

**class** imitation.algorithms.sqil.**SQILReplayBuffer**(*buffer_size*, *observation_space*, *action_space*, *demonstrations*, *device='auto'*, *n_envs=1*, *optimize_memory_usage=False*)

Bases: ReplayBuffer

A replay buffer that injects 50% expert demonstrations when sampling.

This buffer is fundamentally the same as ReplayBuffer, but it includes an expert demonstration internal buffer. When sampling a batch of data, it will be 50/50 expert and collected data.

It can be used in off-policy algorithms like DQN/SAC/TD3.

Here it is used as part of SQIL, where it is used to train a DQN.

**__init__**(*buffer_size*, *observation_space*, *action_space*, *demonstrations*, *device='auto'*, *n_envs=1*, *optimize_memory_usage=False*)

Create a SQILReplayBuffer instance.

**Parameters**

- **buffer_size** (int) – Max number of elements in the buffer
- **observation_space** (Space) – Observation space
- **action_space** (Space) – Action space
- **demonstrations** (Union[Iterable[*Trajectory*], Iterable[*TransitionMapping*], *TransitionsMinimal*]) – Expert demonstrations.
- **device** (Union[device, str]) – PyTorch device.
- **n_envs** (int) – Number of parallel environments. Defaults to 1.
- **optimize_memory_usage** (bool) – Enable a memory efficient variant of the replay buffer which reduces by almost a factor two the memory used, at a cost of more complexity.

**actions: ndarray**

**add**(*obs*, *next_obs*, *action*, *reward*, *done*, *infos*)

Add elements to the buffer.

**Return type**
None

**dones: ndarray**

**next_observations: ndarray**

**observations: ndarray**

**rewards: ndarray**

**sample**(*batch_size*, *env=None*)

Sample a batch of data.

Half of the batch will be from expert transitions, and the other half will be from the learner transitions.

**Parameters**

- **batch_size** (int) – Number of elements to sample in total
- **env** (Optional[VecNormalize]) – associated gym VecEnv to normalize the observations/rewards when sampling

**Return type**

ReplayBufferSamples

**Returns**

A mix of transitions from the expert and from the learner.

**set_demonstrations** (*demonstrations*)

Set the expert demonstrations to be injected when sampling from the buffer.

**Parameters**

**demonstrations** (*algo_base.AnyTransitions*) – Expert demonstrations.

**Raises**

**NotImplementedError** – If *demonstrations* is not a transitions object or a list of trajectories.

**Return type**

None

**timeouts: ndarray**

## 3.1.2 imitation.data

Modules handling environment data.

For example: types for transitions/trajectories; methods to compute rollouts; buffers to store transitions; helpers for these modules.

### Modules

| | |
|---|---|
| *imitation.data.buffer* | Buffers to store NumPy arrays and transitions in. |
| *imitation.data.huggingface_utils* | Helpers to convert between Trajectories and HuggingFace's datasets library. |
| *imitation.data.rollout* | Methods to collect, analyze and manipulate transition and trajectory rollouts. |
| *imitation.data.serialize* | Serialization utilities for trajectories. |
| *imitation.data.types* | Types and helper methods for transitions and trajectories. |
| *imitation.data.wrappers* | Environment wrappers for collecting rollouts. |

### imitation.data.buffer

Buffers to store NumPy arrays and transitions in.

## Functions

| | |
|---|---|
| *num_samples*(data) | Computes the number of samples contained in *data*. |

## Classes

| | |
|---|---|
| *Buffer*(capacity, sample_shapes, dtypes) | A FIFO ring buffer for NumPy arrays of a fixed shape and dtype. |
| *ReplayBuffer*(capacity[, venv, obs_shape, ...]) | Buffer for Transitions. |

**class** `imitation.data.buffer.`**`Buffer`**(*capacity*, *sample_shapes*, *dtypes*)

Bases: `object`

A FIFO ring buffer for NumPy arrays of a fixed shape and dtype.

Supports random sampling with replacement.

**`__init__`**(*capacity*, *sample_shapes*, *dtypes*)

Constructs a Buffer.

**Parameters**

- **capacity** (`int`) – The number of samples that can be stored.
- **sample_shapes** (`Mapping[str, Tuple[int, ...]]`) – A dictionary mapping string keys to the shape of samples associated with that key.
- **dtypes** (*np.dtype*-like) – A dictionary mapping string keys to the dtype of samples associated with that key.

**Raises**

**KeyError** – *sample_shapes* and *dtypes* have different keys.

**capacity: int**

The number of data samples that can be stored in this buffer.

**classmethod from_data**(*data*, *capacity=None*, *truncate_ok=False*)

Constructs and return a Buffer containing the provided data.

Shapes and dtypes are automatically inferred.

**Parameters**

- **data** (`Mapping[str, ndarray]`) – A dictionary mapping keys to data arrays. The arrays may differ in their shape, but should agree in the first axis.
- **capacity** (`Optional[int]`) – The Buffer capacity. If not provided, then this is automatically set to the size of the data, so that the returned Buffer is at full capacity.
- **truncate_ok** (`bool`) – Whether to error if *capacity* < the number of samples in *data*. If False, then only store the last *capacity* samples from *data* when overcapacity.

### Examples

In the follow examples, suppose the arrays in *data* are length-1000.

*Buffer* with same capacity as arrays in *data*:

```
Buffer.from_data(data)
```

*Buffer* with larger capacity than arrays in *data*:

```
Buffer.from_data(data, 10000)
```

*Buffer with smaller capacity than arrays in `data*. Without *truncate_ok=True*, *from_data* will error:

```
Buffer.from_data(data, 5, truncate_ok=True)
```

**Return type**
`Buffer`

**Returns**
Buffer of specified *capacity* containing provided *data*.

**Raises**

- **`ValueError`** – *data* is empty.
- **`ValueError`** – *data* has items mapping to arrays differing in the length of their first axis.

**`sample`**(*n_samples*)

Uniformly sample *n_samples* samples from the buffer with replacement.

**Parameters**
**`n_samples`** (`int`) – The number of samples to randomly sample.

**Returns**

**An array with shape**
*(n_samples) + self.sample_shape*.

**Return type**
samples (np.ndarray)

**Raises**
**`ValueError`** – The buffer is empty.

**`sample_shapes: Mapping[str, Tuple[int, ...]]`**

The shapes of each data sample stored in this buffer.

**`size`**()

Returns the number of samples stored in the buffer.

**Return type**
`int`

**`store`**(*data*, *truncate_ok=False*)

Stores new data samples, replacing old samples with FIFO priority.

**Parameters**

- **`data`** (`Mapping[str, ndarray]`) – A dictionary mapping keys *k* to arrays with shape *(n_samples,) + self.sample_shapes[k]*, where *n_samples* is less than or equal to *self.capacity*.

- **truncate_ok** (bool) – If False, then error if the length of *transitions* is greater than *self.capacity*. Otherwise, store only the final *self.capacity* transitions.

**Raises**

- **ValueError** – *data* is empty.
- **ValueError** – If *n_samples* is greater than *self.capacity*.
- **ValueError** – data is the wrong shape.

**Return type**

None

**class** imitation.data.buffer.**ReplayBuffer**(*capacity*, *venv=None*, ***, *obs_shape=None*, *act_shape=None*, *obs_dtype=None*, *act_dtype=None*)

Bases: object

Buffer for Transitions.

**__init__**(*capacity*, *venv=None*, ***, *obs_shape=None*, *act_shape=None*, *obs_dtype=None*, *act_dtype=None*)

Constructs a ReplayBuffer.

**Parameters**

- **capacity** (int) – The number of samples that can be stored.
- **venv** (Optional[VecEnv]) – The environment whose action and observation spaces can be used to determine the data shapes of the underlying buffers. Mutually exclusive with shape and dtype arguments.
- **obs_shape** (Optional[Tuple[int, ...]]) – The shape of the observation space.
- **act_shape** (Optional[Tuple[int, ...]]) – The shape of the action space.
- **obs_dtype** (Optional[dtype]) – The dtype of the observation space.
- **act_dtype** (Optional[dtype]) – The dtype of the action space.

**Raises**

- **ValueError** – Couldn't infer the observation and action shapes and dtypes from the arguments.
- **ValueError** – Specified both venv and shapes/dtypes.

**capacity: int**

The number of data samples that can be stored in this buffer.

**classmethod from_data**(*transitions*, *capacity=None*, *truncate_ok=False*)

Construct and return a ReplayBuffer containing the provided data.

Shapes and dtypes are automatically inferred, and the returned ReplayBuffer is ready for sampling.

**Parameters**

- **transitions** (*Transitions*) – Transitions to store.
- **capacity** (Optional[int]) – The ReplayBuffer capacity. If not provided, then this is automatically set to the size of the data, so that the returned Buffer is at full capacity.
- **truncate_ok** (bool) – Whether to error if *capacity* < the number of samples in *data*. If False, then only store the last *capacity* samples from *data* when overcapacity.

### Examples

*ReplayBuffer* with same capacity as arrays in *data*:

```
ReplayBuffer.from_data(data)
```

*ReplayBuffer* with larger capacity than arrays in *data*:

```
ReplayBuffer.from_data(data, 10000)
```

*ReplayBuffer with smaller capacity than arrays in `data*. Without *truncate_ok=True*, *from_data* will error:

```
ReplayBuffer.from_data(data, 5, truncate_ok=True)
```

**Return type**
`ReplayBuffer`

**Returns**
A new ReplayBuffer.

**`sample`**(*n_samples*)

Sample obs-act-obs triples.

**Parameters**
**`n_samples`** (`int`) – The number of samples.

**Return type**
`Transitions`

**Returns**
A Transitions named tuple containing n_samples transitions.

**`size`**()

Returns the number of samples stored in the buffer.

**Return type**
`Optional[int]`

**`store`**(*transitions*, *truncate_ok=True*)

Store obs-act-obs triples.

**Parameters**

- **`transitions`** (`Transitions`) – Transitions to store.
- **`truncate_ok`** (`bool`) – If False, then error if the length of *transitions* is greater than *self.capacity*. Otherwise, store only the final *self.capacity* transitions.

**Raises**
**`ValueError`** – The arguments didn't have the same length.

**Return type**
`None`

`imitation.data.buffer.`**`num_samples`**(*data*)

Computes the number of samples contained in *data*.

**Parameters**
**`data`** (`Mapping[Any, ndarray]`) – A Mapping from keys to NumPy arrays.

**Return type**
`int`

**Returns**
The unique length of the first dimension of arrays contained in *data*.

**Raises**
**`ValueError`** – The length is not unique.

### imitation.data.huggingface_utils

Helpers to convert between Trajectories and HuggingFace's datasets library.

#### Functions

| | |
|---|---|
| `trajectories_to_dataset`(trajectories[, info]) | Convert a sequence of trajectories to a HuggingFace dataset. |
| `trajectories_to_dict`(trajectories) | Convert a sequence of trajectories to a dict. |

#### Classes

| | |
|---|---|
| `TrajectoryDatasetSequence`(dataset) | A wrapper to present an HF dataset as a sequence of trajectories. |

**class** `imitation.data.huggingface_utils.`**`TrajectoryDatasetSequence`**(*dataset*)

Bases: `Sequence[Trajectory]`

A wrapper to present an HF dataset as a sequence of trajectories.

Converts the dataset to a sequence of trajectories on the fly.

**`__init__`**(*dataset*)

Construct a TrajectoryDatasetSequence.

**property dataset**

Return the underlying HF dataset.

`imitation.data.huggingface_utils.`**`trajectories_to_dataset`**(*trajectories*, *info=None*)

Convert a sequence of trajectories to a HuggingFace dataset.

**Return type**
`Dataset`

`imitation.data.huggingface_utils.`**`trajectories_to_dict`**(*trajectories*)

Convert a sequence of trajectories to a dict.

The dict has the following fields:

- obs: The observations. Shape: (num_trajectories, num_timesteps, obs_dim).
- acts: The actions. Shape: (num_trajectories, num_timesteps, act_dim).
- infos: The infos. Shape: (num_trajectories, num_timesteps) as jsonpickled str.
- terminal: The terminal flags. Shape: (num_trajectories, num_timesteps, ).

- rews: The rewards. Shape: (num_trajectories, num_timesteps) if applicable.

This dict can be used to construct a HuggingFace dataset.

**Parameters**

**trajectories** (Sequence[*Trajectory*]) – The trajectories to save.

**Raises**

**ValueError** – If not all trajectories have the same type, i.e. some are *Trajectory* and others are *TrajectoryWithRew*.

**Return type**

Dict[str, Sequence[Any]]

**Returns**

A dict representing the trajectories.

### imitation.data.rollout

Methods to collect, analyze and manipulate transition and trajectory rollouts.

#### Functions

| | |
|---|---|
| *discounted_sum*(arr, gamma) | Calculate the discounted sum of *arr*. |
| *flatten_trajectories*(trajectories) | Flatten a series of trajectory dictionaries into arrays. |
| *flatten_trajectories_with_rew*(trajectories) | **rtype**<br>*TransitionsWithRew* |
| *generate_trajectories*(policy, venv, ...[, ...]) | Generate trajectory dictionaries from a policy and an environment. |
| *generate_transitions*(policy, venv, ...[, ...]) | Generate obs-action-next_obs-reward tuples. |
| *make_min_episodes*(n) | Terminate after collecting n episodes of data. |
| *make_min_timesteps*(n) | Terminate at the first episode after collecting n timesteps of data. |
| *make_sample_until*([min_timesteps, min_episodes]) | Returns a termination condition sampling for a number of timesteps and episodes. |
| *policy_to_callable*(policy, venv[, ...]) | Converts any policy-like object into a function from observations to actions. |
| *rollout*(policy, venv, sample_until, rng, *) | Generate policy rollouts. |
| *rollout_stats*(trajectories) | Calculates various stats for a sequence of trajectories. |
| *unwrap_traj*(traj) | Uses *RolloutInfoWrapper*-captured *obs* and *rews* to replace fields. |

### Classes

| | |
|---|---|
| *TrajectoryAccumulator*() | Accumulates trajectories step-by-step. |

**class** imitation.data.rollout.**TrajectoryAccumulator**

Bases: object

Accumulates trajectories step-by-step.

Useful for collecting completed trajectories while ignoring partially-completed trajectories (e.g. when rolling out a VecEnv to collect a set number of transitions). Each in-progress trajectory is identified by a 'key', which enables several independent trajectories to be collected at once. They key can also be left at its default value of *None* if you only wish to collect one trajectory.

**__init__**()

Initialise the trajectory accumulator.

**add_step**(*step_dict*, *key=None*)

Add a single step to the partial trajectory identified by *key*.

Generally a single step could correspond to, e.g., one environment managed by a VecEnv.

**Parameters**

- **step_dict** (Mapping[str, Union[ndarray, *DictObs*, Mapping[str, Any]]]) – dictionary containing information for the current step. Its keys could include any (or all) attributes of a *TrajectoryWithRew* (e.g. "obs", "acts", etc.).
- **key** (Optional[Hashable]) – key to uniquely identify the trajectory to append to, if working with multiple partial trajectories.

**Return type**

None

**add_steps_and_auto_finish**(*acts*, *obs*, *rews*, *dones*, *infos*)

Calls *add_step* repeatedly using acts and the returns from *venv.step*.

Also automatically calls *finish_trajectory()* for each *done == True*. Before calling this method, each environment index key needs to be initialized with the initial observation (usually from *venv.reset()*).

See the body of *util.rollout.generate_trajectory* for an example.

**Parameters**

- **acts** (ndarray) – Actions passed into *VecEnv.step()*.
- **obs** (Union[ndarray, *DictObs*, Dict[str, ndarray]]) – Return value from *VecEnv.step(acts)*.
- **rews** (ndarray) – Return value from *VecEnv.step(acts)*.
- **dones** (ndarray) – Return value from *VecEnv.step(acts)*.
- **infos** (List[dict]) – Return value from *VecEnv.step(acts)*.

**Return type**

List[*TrajectoryWithRew*]

**Returns**

A list of completed trajectories. There should be one trajectory for each *True* in the *dones* argument.

**finish_trajectory**(*key*, *terminal*)

Complete the trajectory labelled with *key*.

**Parameters**

- **key** (Hashable) – key uniquely identifying which in-progress trajectory to remove.
- **terminal** (bool) – trajectory has naturally finished (i.e. includes terminal state).

**Returns**

**list of completed trajectories popped from** *self.partial_trajectories*.

**Return type**

traj

imitation.data.rollout.**discounted_sum**(*arr*, *gamma*)

Calculate the discounted sum of *arr*.

If *arr* is an array of rewards, then this computes the return; however, it can also be used to e.g. compute discounted state occupancy measures.

**Parameters**

- **arr** (ndarray) – 1 or 2-dimensional array to compute discounted sum over. Last axis is timestep, from current time step (first) to last timestep (last). First axis (if present) is batch dimension.
- **gamma** (float) – the discount factor used.

**Return type**

Union[ndarray, float]

**Returns**

The discounted sum over the timestep axis. The first timestep is undiscounted, i.e. we start at gamma^0.

imitation.data.rollout.**flatten_trajectories**(*trajectories*)

Flatten a series of trajectory dictionaries into arrays.

**Parameters**

**trajectories** (Iterable[*Trajectory*]) – list of trajectories.

**Return type**

*Transitions*

**Returns**

The trajectories flattened into a single batch of Transitions.

imitation.data.rollout.**flatten_trajectories_with_rew**(*trajectories*)

**Return type**

*TransitionsWithRew*

imitation.data.rollout.**generate_trajectories**(*policy*, *venv*, *sample_until*, *rng*, ***, *deterministic_policy=False*)

Generate trajectory dictionaries from a policy and an environment.

**Parameters**

- **policy** (Union[BaseAlgorithm, BasePolicy, Callable[[Union[ndarray, Dict[str, ndarray]], Optional[Tuple[ndarray, ...]], Optional[ndarray]], Tuple[ndarray, Optional[Tuple[ndarray, ...]]]], None]) – Can be any of the following: 1) A stable_baselines3 policy or algorithm trained on the gym environment. 2) A Callable that takes an ndarray of observations and returns an ndarray of corresponding actions. 3) None, in which case actions will be sampled randomly.

- **venv** (VecEnv) – The vectorized environments to interact with.
- **sample_until** (Callable[[Sequence[*TrajectoryWithRew*]], bool]) – A function determining the termination condition. It takes a sequence of trajectories, and returns a bool. Most users will want to use one of *min_episodes* or *min_timesteps*.
- **deterministic_policy** (bool) – If True, asks policy to deterministically return action. Note the trajectories might still be non-deterministic if the environment has non-determinism!
- **rng** (Generator) – used for shuffling trajectories.

**Return type**

Sequence[*TrajectoryWithRew*]

**Returns**

Sequence of trajectories, satisfying *sample_until*. Additional trajectories may be collected to avoid biasing process towards short episodes; the user should truncate if required.

imitation.data.rollout.**generate_transitions**(*policy*, *venv*, *n_timesteps*, *rng*, ***, *truncate=True*, ***kwargs*)

Generate obs-action-next_obs-reward tuples.

**Parameters**

- **policy** (Union[BaseAlgorithm, BasePolicy, Callable[[Union[ndarray, Dict[str, ndarray]], Optional[Tuple[ndarray, ...]], Optional[ndarray]], Tuple[ndarray, Optional[Tuple[ndarray, ...]]]], None]) – Can be any of the following: - A stable_baselines3 policy or algorithm trained on the gym environment - A Callable that takes an ndarray of observations and returns an ndarray of corresponding actions - None, in which case actions will be sampled randomly
- **venv** (VecEnv) – The vectorized environments to interact with.
- **n_timesteps** (int) – The minimum number of timesteps to sample.
- **rng** (Generator) – The random state to use for sampling trajectories.
- **truncate** (bool) – If True, then drop any additional samples to ensure that exactly *n_timesteps* samples are returned.
- ****kwargs** – Passed-through to generate_trajectories.

**Return type**

*TransitionsWithRew*

**Returns**

A batch of Transitions. The length of the constituent arrays is guaranteed to be at least *n_timesteps* (if specified), but may be greater unless *truncate* is provided as we collect data until the end of each episode.

imitation.data.rollout.**make_min_episodes**(*n*)

Terminate after collecting n episodes of data.

**Parameters**

**n** (int) – Minimum number of episodes of data to collect. May overshoot if two episodes complete simultaneously (unlikely).

**Return type**

Callable[[Sequence[*TrajectoryWithRew*]], bool]

**Returns**
A function implementing this termination condition.

imitation.data.rollout.**make_min_timesteps**(*n*)

Terminate at the first episode after collecting n timesteps of data.

**Parameters**
**n** (int) – Minimum number of timesteps of data to collect. May overshoot to nearest episode boundary.

**Return type**
Callable[[Sequence[*TrajectoryWithRew*]], bool]

**Returns**
A function implementing this termination condition.

imitation.data.rollout.**make_sample_until**(*min_timesteps=None*, *min_episodes=None*)

Returns a termination condition sampling for a number of timesteps and episodes.

**Parameters**

- **min_timesteps** (Optional[int]) – Sampling will not stop until there are at least this many timesteps.
- **min_episodes** (Optional[int]) – Sampling will not stop until there are at least this many episodes.

**Return type**
Callable[[Sequence[*TrajectoryWithRew*]], bool]

**Returns**
A termination condition.

**Raises**
**ValueError** – Neither of n_timesteps and n_episodes are set, or either are non-positive.

imitation.data.rollout.**policy_to_callable**(*policy*, *venv*, *deterministic_policy=False*)

Converts any policy-like object into a function from observations to actions.

**Return type**
Callable[[Union[ndarray, Dict[str, ndarray]], Optional[Tuple[ndarray, ...]], Optional[ndarray]], Tuple[ndarray, Optional[Tuple[ndarray, ...]]]]

imitation.data.rollout.**rollout**(*policy*, *venv*, *sample_until*, *rng*, ***, *unwrap=True*, *exclude_infos=True*, *verbose=True*, ***kwargs*)

Generate policy rollouts.

This method is a wrapper of generate_trajectories that allows the user to additionally replace the rewards and observations with the original values if the environment is wrapped, to exclude the infos from the trajectories, and to print summary statistics of the rollout.

The *.infos* field of each Trajectory is set to *None* to save space.

**Parameters**

- **policy** (Union[BaseAlgorithm, BasePolicy, Callable[[Union[ndarray, Dict[str, ndarray]], Optional[Tuple[ndarray, ...]], Optional[ndarray]], Tuple[ndarray, Optional[Tuple[ndarray, ...]]]], None]) – Can be any of the following: 1) A stable_baselines3 policy or algorithm trained on the gym environment. 2) A Callable that takes an ndarray of observations and returns an ndarray of corresponding actions. 3) None, in which case actions will be sampled randomly.

- **venv** (VecEnv) – The vectorized environments.
- **sample_until** (Callable[[Sequence[*TrajectoryWithRew*]], bool]) – End condition for rollout sampling.
- **rng** (Generator) – Random state to use for sampling.
- **unwrap** (bool) – If True, then save original observations and rewards (instead of potentially wrapped observations and rewards) by calling *unwrap_traj()*.
- **exclude_infos** (bool) – If True, then exclude *infos* from pickle by setting this field to None. Excluding *infos* can save a lot of space during pickles.
- **verbose** (bool) – If True, then print out rollout stats before saving.
- ****kwargs** – Passed through to *generate_trajectories*.

**Return type**
Sequence[*TrajectoryWithRew*]

**Returns**
Sequence of trajectories, satisfying *sample_until*. Additional trajectories may be collected to avoid biasing process towards short episodes; the user should truncate if required.

imitation.data.rollout.**rollout_stats**(*trajectories*)

Calculates various stats for a sequence of trajectories.

**Parameters**
**trajectories** (Sequence[*TrajectoryWithRew*]) – Sequence of trajectories.

**Return type**
Mapping[str, float]

**Returns**

Dictionary containing *n_traj* collected (int), along with episode return statistics (keys: *{monitor_,}return_{min,mean,std,max}*, float values) and trajectory length statistics (keys: *len_{min,mean,std,max}*, float values).

*return_** values are calculated from environment rewards. *monitor_** values are calculated from Monitor-captured rewards, and are only included if the *trajectories* contain Monitor infos.

imitation.data.rollout.**unwrap_traj**(*traj*)

Uses *RolloutInfoWrapper*-captured *obs* and *rews* to replace fields.

This can be useful for bypassing other wrappers to retrieve the original *obs* and *rews*.

Fails if *infos* is None or if the trajectory was generated from an environment without imitation.data.wrappers.RolloutInfoWrapper

**Parameters**
**traj** (*TrajectoryWithRew*) – A trajectory generated from *RolloutInfoWrapper*-wrapped Environments.

**Return type**
*TrajectoryWithRew*

**Returns**
A copy of *traj* with replaced *obs* and *rews* fields.

**Raises**
**ValueError** – If *traj.infos* is None

### imitation.data.serialize

Serialization utilities for trajectories.

#### Functions

| | |
|---|---|
| *load*(path) | Loads a sequence of trajectories saved by *save()* from *path*. |
| *load_with_rewards*(path) | Loads a sequence of trajectories with rewards from a file. |
| *save*(path, trajectories) | Save a sequence of Trajectories to disk using HuggingFace's datasets library. |

imitation.data.serialize.**load**(*path*)

Loads a sequence of trajectories saved by *save()* from *path*.

**Return type**

Sequence[*Trajectory*]

imitation.data.serialize.**load_with_rewards**(*path*)

Loads a sequence of trajectories with rewards from a file.

**Return type**

Sequence[*TrajectoryWithRew*]

imitation.data.serialize.**save**(*path*, *trajectories*)

Save a sequence of Trajectories to disk using HuggingFace's datasets library.

**Parameters**

- **path** (Union[str, bytes, PathLike]) – Trajectories are saved to this path.
- **trajectories** (Sequence[*Trajectory*]) – The trajectories to save.

**Return type**

None

### imitation.data.types

Types and helper methods for transitions and trajectories.

### Functions

| | |
|---|---|
| assert_not_dictobs(x) | Typeguard to assert *x* is an array, not a DictObs. |
| concatenate_maybe_dictobs(arrs) | Concatenates a list of observations appropriately (depending on type). |
| dataclass_quick_asdict(obj) | Extract dataclass to items using *dataclasses.fields* + dict comprehension. |
| map_maybe_dict(fn, maybe_dict) | Either maps fn over dictionary values or applies fn to *maybe_dict*. |
| maybe_unwrap_dictobs() | Unwraps if a DictObs, otherwise returns the object. |
| maybe_wrap_in_dictobs() | Converts an observation into a DictObs, if necessary. |
| stack_maybe_dictobs(arrs) | Stacks a list of observations appropriately (depending on type). |
| transitions_collate_fn(batch) | Custom *torch.utils.data.DataLoader* collate_fn for *TransitionsMinimal*. |

### Classes

| | |
|---|---|
| DictObs(_d) | Stores observations from an environment with a dictionary observation space. |
| Trajectory(obs, acts, infos, terminal) | A trajectory, e.g. |
| TrajectoryWithRew(obs, acts, infos, ...) | A *Trajectory* that additionally includes reward information. |
| TransitionMapping(*args, **kwargs) | Dictionary with *obs* and *acts*, maybe also *next_obs*, *dones*, *rew*. |
| TransitionMappingNoNextObs(*args, **kwargs) | Dictionary with *obs* and *acts*. |
| Transitions(obs, acts, infos, next_obs, dones) | A batch of obs-act-obs-done transitions. |
| TransitionsMinimal(obs, acts, infos) | A Torch-compatible *Dataset* of obs-act transitions. |
| TransitionsWithRew(obs, acts, infos, ...) | A batch of obs-act-obs-rew-done transitions. |

**class** imitation.data.types.**DictObs**(*_d*)

Bases: `object`

Stores observations from an environment with a dictionary observation space.

Provides an interface that is similar to observations in a numpy array. Length, slicing, indexing, and iterating operations will operate on the first dimension of the constituent arrays, as they would for observations in a single array.

There are also utility functions for mapping / stacking / concatenating lists of dictobs.

**__init__**(*_d*)

**classmethod concatenate**(*dictobs_list*, *axis=0*)

Returns a single dictobs concatenating the arrays by key.

**Return type**

DictObs

**property dict_len**

Returns the number of arrays in the DictObs.

**property dtype: Dict[str, dtype]**

Returns a dictionary with dtype-tuples in place of the arrays.

**Return type**

`Dict[str, dtype]`

**classmethod from_obs_list**(*obs_list*)

Stacks the observation list into a single DictObs.

**Return type**

`DictObs`

**get**(*key*)

Returns the array for the given key, or raises KeyError.

**Return type**

`ndarray`

**items**()

**keys**()

**map_arrays**(*fn*)

Returns a new DictObs with *fn* applied to every array.

**Return type**

`DictObs`

**property shape: Dict[str, Tuple[int, ...]]**

Returns a dictionary with shape-tuples in place of the arrays.

**Return type**

`Dict[str, Tuple[int, ...]]`

**classmethod stack**(*dictobs_list*, *axis=0*)

Returns a single dictobs stacking the arrays by key.

**Return type**

`DictObs`

**unwrap**()

Returns a copy of the underlying dictionary (arrays are not copied).

**Return type**

`Dict[str, ndarray]`

**values**()

**class** imitation.data.types.**Trajectory**(*obs*, *acts*, *infos*, *terminal*)

Bases: `object`

A trajectory, e.g. a one episode rollout from an expert policy.

**__init__**(*obs*, *acts*, *infos*, *terminal*)

**acts: ndarray**

Actions, shape (trajectory_len, ) + action_shape.

**infos: Optional[ndarray]**

An array of info dicts, shape (trajectory_len, ).

The info dict is returned by some environments *step()* and contains auxiliary diagnostic information. For example the monitor wrapper adds an info dict to the last step of each episode containing the episode return and length.

**obs: Union[ndarray, *DictObs*]**

Observations, shape (trajectory_len + 1, ) + observation_shape.

**terminal: bool**

Does this trajectory (fragment) end in a terminal state?

Episodes are always terminal. Trajectory fragments are also terminal when they contain the final state of an episode (even if missing the start of the episode).

**class** `imitation.data.types.`**TrajectoryWithRew**(*obs*, *acts*, *infos*, *terminal*, *rews*)

Bases: *Trajectory*

A *Trajectory* that additionally includes reward information.

**__init__**(*obs*, *acts*, *infos*, *terminal*, *rews*)

**rews: ndarray**

Reward, shape (trajectory_len, ). dtype float.

**class** `imitation.data.types.`**TransitionMapping**(**args*, ***kwargs*)

Bases: `dict`

Dictionary with *obs* and *acts*, maybe also *next_obs*, *dones*, *rew*.

**acts: Union[ndarray, Tensor]**

**dones: Union[ndarray, Tensor]**

**next_obs: Union[ndarray, *DictObs*, Tensor]**

**obs: Union[ndarray, *DictObs*, Tensor]**

**rew: Union[ndarray, Tensor]**

**class** `imitation.data.types.`**TransitionMappingNoNextObs**(**args*, ***kwargs*)

Bases: `dict`

Dictionary with *obs* and *acts*.

**acts: Union[ndarray, Tensor]**

**obs: Union[ndarray, *DictObs*, Tensor]**

**class** `imitation.data.types.`**Transitions**(*obs*, *acts*, *infos*, *next_obs*, *dones*)

Bases: *TransitionsMinimal*

A batch of obs-act-obs-done transitions.

**__init__**(*obs*, *acts*, *infos*, *next_obs*, *dones*)

**dones: ndarray**

(batch_size, ).

*done[i]* is true iff *next_obs[i]* the last observation of an episode.

**Type**

Boolean array indicating episode termination. Shape

**next_obs: Union[ndarray, *DictObs*]**

(batch_size, ) + observation_shape.

The i'th observation *next_obs[i]* in this array is the observation after the agent has taken action *acts[i]*.

**Invariants:**

- *next_obs.dtype == obs.dtype*
- *len(next_obs) == len(obs)*

**Type**

New observation. Shape

**class** `imitation.data.types.`**TransitionsMinimal**(*obs*, *acts*, *infos*)

Bases: `Dataset`, `Sequence[Mapping[str, ndarray]]`

A Torch-compatible *Dataset* of obs-act transitions.

This class and its subclasses are usually instantiated via *imitation.data.rollout.flatten_trajectories*.

Indexing an instance *trans* of TransitionsMinimal with an integer *i* returns the *i`th `Dict[str, np.ndarray]* sample, whose keys are the field names of each dataclass field and whose values are the ith elements of each field value.

Slicing returns a possibly empty instance of *TransitionsMinimal* where each field has been sliced.

**__init__**(*obs*, *acts*, *infos*)

**acts: ndarray**

(batch_size,) + action_shape.

**Type**

Actions. Shape

**infos: ndarray**

(batch_size,).

**Type**

Array of info dicts. Shape

**obs: Union[ndarray, *DictObs*]**

(batch_size, ) + observation_shape.

The i'th observation *obs[i]* in this array is the observation seen by the agent when choosing action *acts[i]*. *obs[i]* is not required to be from the timestep preceding *obs[i+1]*.

**Type**

Previous observations. Shape

**class** `imitation.data.types.`**TransitionsWithRew**(*obs*, *acts*, *infos*, *next_obs*, *dones*, *rews*)

Bases: *`Transitions`*

A batch of obs-act-obs-rew-done transitions.

**__init__**(*obs*, *acts*, *infos*, *next_obs*, *dones*, *rews*)

**rews: ndarray**

(batch_size, ). dtype float.

The reward *rew[i]* at the i'th timestep is received after the agent has taken action *acts[i]*.

**Type**

Reward. Shape

`imitation.data.types.`**`assert_not_dictobs`**`(`*x*`)`

Typeguard to assert *x* is an array, not a DictObs.

**Return type**

`ndarray`

`imitation.data.types.`**`concatenate_maybe_dictobs`**`(`*arrs*`)`

Concatenates a list of observations appropriately (depending on type).

**Return type**

`TypeVar`(`ObsVar`, `ndarray`, *`DictObs`*)

`imitation.data.types.`**`dataclass_quick_asdict`**`(`*obj*`)`

Extract dataclass to items using *dataclasses.fields* + dict comprehension.

This is a quick alternative to *dataclasses.asdict*, which expensively and undocumentedly deep-copies every numpy array value. See https://stackoverflow.com/a/52229565/1091722.

This is also used to preserve DictObj objects, as *dataclasses.asdict* unwraps them recursively.

**Parameters**

**`obj`** – A dataclass instance.

**Return type**

`Dict`[`str`, `Any`]

**Returns**

A dictionary mapping from *obj* field names to values.

`imitation.data.types.`**`map_maybe_dict`**`(`*fn*, *maybe_dict*`)`

Either maps fn over dictionary values or applies fn to *maybe_dict*.

**Parameters**

- **`fn`** – function to apply. Must take a single argument.
- **`maybe_dict`** – either a dict or a value that can be passed to fn.

**Returns**

Either a dict (if maybe_dict was a dict) or *fn(maybe_dict)*.

`imitation.data.types.`**`maybe_unwrap_dictobs`**`(`*maybe_dictobs:* DictObs`)` → Dict[str, ndarray]

`imitation.data.types.`**`maybe_unwrap_dictobs`**`(`*maybe_dictobs: T*`)` → T

Unwraps if a DictObs, otherwise returns the object.

`imitation.data.types.`**`maybe_wrap_in_dictobs`**`(`*obs: Union[Dict[str, ndarray],* DictObs*]*`)` → *DictObs*

`imitation.data.types.`**`maybe_wrap_in_dictobs`**`(`*obs: ndarray*`)` → ndarray

Converts an observation into a DictObs, if necessary.

**Return type**

`Union`[`ndarray`, *`DictObs`*]

imitation.data.types.**stack_maybe_dictobs**(*arrs*)

Stacks a list of observations appropriately (depending on type).

**Return type**

TypeVar(ObsVar, ndarray, *DictObs*)

imitation.data.types.**transitions_collate_fn**(*batch*)

Custom *torch.utils.data.DataLoader* collate_fn for *TransitionsMinimal*.

Use this as the *collate_fn* argument to *DataLoader* if using an instance of *TransitionsMinimal* as the *dataset* argument.

**Parameters**

**batch** (Sequence[Mapping[str, ndarray]]) – The batch to collate.

**Return type**

Mapping[str, Union[ndarray, Tensor]]

**Returns**

A collated batch. Uses Torch's default collate function for everything except the "infos" key. For "infos", we join all the info dicts into a list of dicts. (The default behavior would recursively collate every info dict into a single dict, which is incorrect.)

### imitation.data.wrappers

Environment wrappers for collecting rollouts.

### Classes

| | |
|---|---|
| *BufferingWrapper*(venv[, ...]) | Saves transitions of underlying VecEnv. |
| *RolloutInfoWrapper*(env) | Add the entire episode's rewards and observations to *info* at episode end. |

**class** imitation.data.wrappers.**BufferingWrapper**(*venv*, *error_on_premature_reset=True*)

Bases: VecEnvWrapper

Saves transitions of underlying VecEnv.

Retrieve saved transitions using *pop_transitions()*.

**__init__**(*venv*, *error_on_premature_reset=True*)

Builds BufferingWrapper.

**Parameters**

- **venv** (VecEnv) – The wrapped VecEnv.
- **error_on_premature_reset** (bool) – Error if *reset()* is called on this wrapper and there are saved samples that haven't yet been accessed.

**error_on_premature_event: bool**

**n_transitions: Optional[int]**

**pop_finished_trajectories**()

Pops recorded complete trajectories *trajs* and episode lengths *ep_lens*.

**Return type**
`Tuple[Sequence[`*`TrajectoryWithRew`*`], Sequence[int]]`

**Returns**
A tuple *(trajs, ep_lens)* where *trajs* is a sequence of trajectories including the terminal state (but possibly missing initial states, if *pop_trajectories* was previously called) and *ep_lens* is a sequence of episode lengths. Note the episode length will be longer than the trajectory length when the trajectory misses initial states.

**pop_trajectories**()

Pops recorded trajectories *trajs* and episode lengths *ep_lens*.

**Return type**
`Tuple[Sequence[`*`TrajectoryWithRew`*`], Sequence[int]]`

**Returns**
A tuple *(trajs, ep_lens)*. *trajs* is a sequence of trajectory fragments, consisting of data collected after the last call to *pop_trajectories*. They may miss initial states (if *pop_trajectories* previously returned a fragment for that episode), and terminal states (if the episode has yet to complete). *ep_lens* is the total length of completed episodes.

**pop_transitions**()

Pops recorded transitions, returning them as an instance of Transitions.

**Return type**
*`TransitionsWithRew`*

**Returns**
All transitions recorded since the last call.

**Raises**
**RuntimeError** – empty (no transitions recorded since last pop).

**reset**(***kwargs*)

Reset all the environments and return an array of observations, or a tuple of observation arrays.

If step_async is still doing work, that work will be cancelled and step_wait() should not be called until step_async() is invoked again.

**Returns**
observation

**step_async**(*actions*)

Tell all the environments to start taking a step with the given actions. Call step_wait() to get the results of the step.

You should not call this if a step_async run is already pending.

**step_wait**()

Wait for the step taken with step_async().

**Returns**
observation, reward, done, information

**class** `imitation.data.wrappers.`**RolloutInfoWrapper**(*env*)

Bases: `Wrapper`

Add the entire episode's rewards and observations to *info* at episode end.

Whenever done=True, *info["rollouts"]* is a dict with keys "obs" and "rews", whose corresponding values hold the NumPy arrays containing the raw observations and rewards seen during this episode.

**`__init__`**(*env*)

Builds RolloutInfoWrapper.

**Parameters**
**env** (`Env`) – Environment to wrap.

**`reset`**(***kwargs*)

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

**`step`**(*action*)

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

## 3.1.3 imitation.policies

Classes defining policies and methods to manipulate them (e.g. serialization).

### Modules

| | |
|---|---|
| *imitation.policies.base* | Custom policy classes and convenience methods. |
| *imitation.policies.exploration_wrapper* | Wrapper to turn a policy into a more exploratory version. |
| *imitation.policies.interactive* | Interactive policies that query the user for actions. |
| *imitation.policies.replay_buffer_wrapper* | Wrapper for reward labeling for transitions sampled from a replay buffer. |
| *imitation.policies.serialize* | Load serialized policies of different types. |

### imitation.policies.base

Custom policy classes and convenience methods.

### Classes

| | |
|---|---|
| *FeedForward32Policy*(*args, **kwargs) | A feed forward policy network with two hidden layers of 32 units. |
| *NonTrainablePolicy*(observation_space, ...) | Abstract class for non-trainable (e.g. |
| *NormalizeFeaturesExtractor*(observation_space) | Feature extractor that flattens then normalizes input. |
| *RandomPolicy*(observation_space, action_space) | Returns random actions. |
| *SAC1024Policy*(*args, **kwargs) | Actor and value networks with two hidden layers of 1024 units respectively. |
| *ZeroPolicy*(observation_space, action_space) | Returns constant zero action. |

**class** `imitation.policies.base.`**`FeedForward32Policy`**(**args*, ***kwargs*)

Bases: `ActorCriticPolicy`

A feed forward policy network with two hidden layers of 32 units.

This matches the IRL policies in the original AIRL paper.

Note: This differs from stable_baselines3 ActorCriticPolicy in two ways: by having 32 rather than 64 units, and by having policy and value networks share weights except at the final layer, where there are different linear heads.

**`__init__`**(**args*, ***kwargs*)

Builds FeedForward32Policy; arguments passed to *ActorCriticPolicy*.

**`features_extractor: BaseFeaturesExtractor`**

**class** `imitation.policies.base.`**`NonTrainablePolicy`**(*observation_space*, *action_space*)

Bases: `BasePolicy`, `ABC`

Abstract class for non-trainable (e.g. hard-coded or interactive) policies.

**`__init__`**(*observation_space*, *action_space*)

Builds NonTrainablePolicy with specified observation and action space.

**`features_extractor: BaseFeaturesExtractor`**

**`forward`**(**args*)

Define 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.

**class** `imitation.policies.base.`**`NormalizeFeaturesExtractor`**(*observation_space*, *normalize_class=<class 'imitation.util.networks.RunningNorm'>*)

Bases: `FlattenExtractor`

Feature extractor that flattens then normalizes input.

**`__init__`**(*observation_space*, *normalize_class=<class 'imitation.util.networks.RunningNorm'>*)

Builds NormalizeFeaturesExtractor.

**Parameters**

- **`observation_space`** (`Space`) – The space observations lie in.
- **`normalize_class`** (`Type`[`Module`]) – The class to use to normalize observations (after being flattened). This can be any Module that preserves the shape; e.g. *nn.BatchNorm** or *nn.LayerNorm*.

**`forward`**(*observations*)

Define 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.

**Return type**
`Tensor`

**training: bool**

**class** `imitation.policies.base.`**RandomPolicy**(*observation_space*, *action_space*)

Bases: *NonTrainablePolicy*

Returns random actions.

**features_extractor: BaseFeaturesExtractor**

**optimizer: th.optim.Optimizer**

**training: bool**

**class** `imitation.policies.base.`**SAC1024Policy**(**args*, ***kwargs*)

Bases: `SACPolicy`

Actor and value networks with two hidden layers of 1024 units respectively.

This matches the implementation of SAC policies in the PEBBLE paper. See: https://arxiv.org/pdf/2106.05091.pdf https://github.com/denisyarats/pytorch_sac/blob/master/config/agent/sac.yaml

Note: This differs from stable_baselines3 SACPolicy by having 1024 hidden units in each layer instead of the default value of 256.

**__init__**(**args*, ***kwargs*)

Builds SAC1024Policy; arguments passed to *SACPolicy*.

**actor: Actor**

**critic: ContinuousCritic**

**critic_target: ContinuousCritic**

**class** `imitation.policies.base.`**ZeroPolicy**(*observation_space*, *action_space*)

Bases: *NonTrainablePolicy*

Returns constant zero action.

**__init__**(*observation_space*, *action_space*)

Builds ZeroPolicy with specified observation and action space.

**features_extractor: BaseFeaturesExtractor**

**optimizer: th.optim.Optimizer**

**training: bool**

### imitation.policies.exploration_wrapper

Wrapper to turn a policy into a more exploratory version.

### Classes

| | |
|---|---|
| *ExplorationWrapper*(policy, venv, ...[, ...]) | Wraps a PolicyCallable to create a partially randomized version. |

**class** imitation.policies.exploration_wrapper.**ExplorationWrapper**(*policy*, *venv*, *random_prob*, *switch_prob*, *rng*, *deterministic_policy=False*)

Bases: `object`

Wraps a PolicyCallable to create a partially randomized version.

This wrapper randomly switches between two policies: the wrapped policy, and a random one. After each action, the current policy is kept with a certain probability. Otherwise, one of these two policies is chosen at random (without any dependence on what the current policy is).

The random policy uses the *action_space.sample()* method.

**__init__**(*policy*, *venv*, *random_prob*, *switch_prob*, *rng*, *deterministic_policy=False*)

Initializes the ExplorationWrapper.

**Parameters**

- **policy** (`Union[BaseAlgorithm, BasePolicy, Callable[[Union[ndarray, Dict[str, ndarray]], Optional[Tuple[ndarray, ...]], Optional[ndarray]], Tuple[ndarray, Optional[Tuple[ndarray, ...]]]], None]`) – The policy to randomize.
- **venv** (`VecEnv`) – The environment to use (needed for sampling random actions).
- **random_prob** (`float`) – The probability of picking the random policy when switching.
- **switch_prob** (`float`) – The probability of switching away from the current policy.
- **rng** (`Generator`) – The random state to use for seeding the environment and for switching policies.
- **deterministic_policy** (`bool`) – Whether to make the policy deterministic when not exploring. This must be False when `policy` is a `PolicyCallable`.

## imitation.policies.interactive

Interactive policies that query the user for actions.

### Classes

| | |
|---|---|
| *AtariInteractivePolicy*(env, *args, **kwargs) | Interactive policy for Atari environments. |
| *DiscreteInteractivePolicy*(observation_space, ...) | Abstract class for interactive policies with discrete actions. |
| *ImageObsDiscreteInteractivePolicy*(...[, ...]) | DiscreteInteractivePolicy that renders image observations. |

**class** imitation.policies.interactive.**AtariInteractivePolicy**(*env*, **args*, ***kwargs*)

Bases: *ImageObsDiscreteInteractivePolicy*

Interactive policy for Atari environments.

**__init__**(*env*, **args*, ***kwargs*)

Builds AtariInteractivePolicy.

**features_extractor: BaseFeaturesExtractor**

**optimizer: th.optim.Optimizer**

**training: bool**

**class** imitation.policies.interactive.**DiscreteInteractivePolicy**(*observation_space*, *action_space*, *action_keys_names*, *clear_screen_on_query=True*)

Bases: *NonTrainablePolicy*, ABC

Abstract class for interactive policies with discrete actions.

For each query, the observation is rendered and then the action is provided as a keyboard input.

**__init__**(*observation_space*, *action_space*, *action_keys_names*, *clear_screen_on_query=True*)

Builds DiscreteInteractivePolicy.

**Parameters**

- **observation_space** (Space) – Observation space.
- **action_space** (Space) – Action space.
- **action_keys_names** (OrderedDict) – *OrderedDict* containing pairs (key, name) for every action, where key will be used in the console interface, and name is a semantic action name. The index of the pair in the dictionary will be used as the discrete, integer action.
- **clear_screen_on_query** (bool) – If *True*, console will be cleared on every query.

**features_extractor: BaseFeaturesExtractor**

**optimizer: th.optim.Optimizer**

**training: bool**

**class** imitation.policies.interactive.**ImageObsDiscreteInteractivePolicy**(*observation_space*, *action_space*, *action_keys_names*, *clear_screen_on_query=True*

Bases: *DiscreteInteractivePolicy*

DiscreteInteractivePolicy that renders image observations.

**features_extractor: BaseFeaturesExtractor**

**optimizer: th.optim.Optimizer**

**training: bool**

### imitation.policies.replay_buffer_wrapper

Wrapper for reward labeling for transitions sampled from a replay buffer.

#### Classes

| | |
|---|---|
| *ReplayBufferRewardWrapper*(buffer_size, ...) | Relabel the rewards in transitions sampled from a ReplayBuffer. |

**class** imitation.policies.replay_buffer_wrapper.**ReplayBufferRewardWrapper**(*buffer_size*, *observation_space*, *action_space*, ***, *replay_buffer_class*, *reward_fn*, ***kwargs*)

Bases: ReplayBuffer

Relabel the rewards in transitions sampled from a ReplayBuffer.

**__init__**(*buffer_size*, *observation_space*, *action_space*, ***, *replay_buffer_class*, *reward_fn*, ***kwargs*)

Builds ReplayBufferRewardWrapper.

**Parameters**

- **buffer_size** (int) – Max number of elements in the buffer
- **observation_space** (Space) – Observation space
- **action_space** (Space) – Action space
- **replay_buffer_class** (Type[ReplayBuffer]) – Class of the replay buffer.
- **reward_fn** (*RewardFn*) – Reward function for reward relabeling.
- ****kwargs** – keyword arguments for ReplayBuffer.

**actions: ndarray**

**add**(**args*, ***kwargs*)

Add elements to the buffer.

**dones: ndarray**

**property full: bool**

**Return type**

bool

**next_observations: ndarray**

**observations: ndarray**

**property pos: int**

**Return type**
int

**rewards: ndarray**

**sample**(**args*, ***kwargs*)

Sample elements from the replay buffer. Custom sampling when using memory efficient variant, as we should not sample the element with index *self.pos* See https://github.com/DLR-RM/stable-baselines3/pull/28#issuecomment-637559274

**Parameters**

- **batch_size** – Number of element to sample
- **env** – associated gym VecEnv to normalize the observations/rewards when sampling

**Returns**

**timeouts: ndarray**

### imitation.policies.serialize

Load serialized policies of different types.

#### Module Attributes

| | |
|---|---|
| *PolicyLoaderFn* | A policy loader function that takes a VecEnv before any other custom arguments and returns a stable_baselines3 base policy policy. |
| *policy_registry* | Registry of policy loading functions. |

#### Functions

| | |
|---|---|
| *load_policy*(policy_type, venv, **kwargs) | Load serialized policy. |
| *load_stable_baselines_model*(cls, path, venv, ...) | Helper method to load RL models from Stable Baselines. |
| *save_stable_model*(output_dir, model[, filename]) | Serialize Stable Baselines model. |

#### Classes

| | |
|---|---|
| *SavePolicyCallback*(policy_dir, *args, **kwargs) | Saves the policy using *save_stable_model* each time it is called. |

imitation.policies.serialize.**PolicyLoaderFn**

A policy loader function that takes a VecEnv before any other custom arguments and returns a stable_baselines3 base policy policy.

alias of `Callable`[[…], `BasePolicy`]

**class** imitation.policies.serialize.**SavePolicyCallback**(*policy_dir*, **args*, ***kwargs*)

Bases: `EventCallback`

Saves the policy using *save_stable_model* each time it is called.

Should be used in conjunction with *callbacks.EveryNTimesteps* or another event-based trigger.

**__init__**(*policy_dir*, **args*, ***kwargs*)

Builds SavePolicyCallback.

**Parameters**

- **policy_dir** (`Path`) – Directory to save checkpoints.
- ***args** – Passed through to *callbacks.EventCallback*.
- ****kwargs** – Passed through to *callbacks.EventCallback*.

**model: base_class.BaseAlgorithm**

imitation.policies.serialize.**load_policy**(*policy_type*, *venv*, ***kwargs*)

Load serialized policy.

Note on the kwargs:

- *zero* and *random* policy take no kwargs
- *ppo* and *sac* policies take a *path* argument with a path to a zip file or to a folder containing a *model.zip* file.
- *ppo-huggingface* and *sac-huggingface* policies take an *env_name* and optional *organization* argument.

**Parameters**

- **policy_type** (`str`) – A key in *policy_registry*, e.g. *ppo*.
- **venv** (`VecEnv`) – An environment that the policy is to be used with.
- ****kwargs** – Additional arguments to pass to the policy loader.

**Return type**

`BasePolicy`

**Returns**

The deserialized policy.

imitation.policies.serialize.**load_stable_baselines_model**(*cls*, *path*, *venv*, ***kwargs*)

Helper method to load RL models from Stable Baselines.

**Parameters**

- **cls** (`Type`[`TypeVar`(`Algorithm`, bound= `BaseAlgorithm`)]) – Stable Baselines RL algorithm.
- **path** (`str`) – Path to zip file containing saved model data or to a folder containing a *model.zip* file.
- **venv** (`VecEnv`) – Environment to train on.
- **kwargs** – Passed through to *cls.load*.

**Raises**

- **FileNotFoundError** – If *path* is not a directory containing a *model.zip* file.
- **FileExistsError** – If *path* contains a *vec_normalize.pkl* file (unsupported).

**Return type**

`TypeVar`(`Algorithm`, bound= `BaseAlgorithm`)

**Returns**

The deserialized RL algorithm.

`imitation.policies.serialize.`**`policy_registry: Registry[Callable[[...], BasePolicy]] = <imitation.util.registry.Registry object>`**

Registry of policy loading functions. Add your own here if desired.

`imitation.policies.serialize.`**`save_stable_model`**(*output_dir*, *model*, *filename='model.zip'*)

Serialize Stable Baselines model.

Load later with *load_policy(…, policy_path=output_dir)*.

**Parameters**

- **`output_dir`** (`Path`) – Path to the save directory.
- **`model`** (`BaseAlgorithm`) – The stable baselines model.
- **`filename`** (`str`) – The filename of the model.

**Return type**

`None`

## 3.1.4 imitation.regularization

Implements a variety of regularization techniques for NN weights.

### Modules

| | |
|---|---|
| *imitation.regularization.regularizers* | Implements the regularizer base class and some standard regularizers. |
| *imitation.regularization.updaters* | Implements parameter scaling algorithms to update the parameters of a regularizer. |

### imitation.regularization.regularizers

Implements the regularizer base class and some standard regularizers.

### Classes

| | |
|---|---|
| *LossRegularizer*(optimizer, initial_lambda, ...) | Abstract base class for regularizers that add a loss term to the loss function. |
| *LpRegularizer*(optimizer, initial_lambda, ...) | Applies Lp regularization to a loss function. |
| *Regularizer*(optimizer, initial_lambda, ...) | Abstract class for creating regularizers with a common interface. |
| *RegularizerFactory*(*args, **kwargs) | Protocol for functions that create regularizers. |
| *WeightDecayRegularizer*(optimizer, ...[, ...]) | Applies weight decay to a loss function. |
| *WeightRegularizer*(optimizer, initial_lambda, ...) | Abstract base class for regularizers that regularize the weights of a network. |

**class** imitation.regularization.regularizers.**LossRegularizer**(*optimizer*, *initial_lambda*, *lambda_updater*, *logger*, *val_split=None*)

Bases: *Regularizer*[Union[Tensor, float]]

Abstract base class for regularizers that add a loss term to the loss function.

Requires the user to implement the _loss_penalty method.

**lambda_: float**

**lambda_updater: Optional[*LambdaUpdater*]**

**logger: *HierarchicalLogger***

**optimizer: Optimizer**

**regularize_and_backward**(*loss*)

Add the regularization term to the loss and compute gradients.

**Parameters**

**loss** (Tensor) – The loss to regularize.

**Return type**

Union[Tensor, float]

**Returns**

The regularized loss.

**val_split: Optional[float]**

**class** imitation.regularization.regularizers.**LpRegularizer**(*optimizer*, *initial_lambda*, *lambda_updater*, *logger*, *p*, *val_split=None*)

Bases: *LossRegularizer*

Applies Lp regularization to a loss function.

**__init__**(*optimizer*, *initial_lambda*, *lambda_updater*, *logger*, *p*, *val_split=None*)

Initialize the regularizer.

**p: int**

**class** imitation.regularization.regularizers.**Regularizer**(*optimizer*, *initial_lambda*, *lambda_updater*, *logger*, *val_split=None*)

Bases: ABC, Generic[R]

Abstract class for creating regularizers with a common interface.

**__init__**(*optimizer*, *initial_lambda*, *lambda_updater*, *logger*, *val_split=None*)

Initialize the regularizer.

**Parameters**

- **optimizer** (Optimizer) – The optimizer to which the regularizer is attached.
- **initial_lambda** (float) – The initial value of the regularization parameter.
- **lambda_updater** (Optional[*LambdaUpdater*]) – A callable object that takes in the current lambda and the train and val loss, and returns the new lambda.

- **logger** (*HierarchicalLogger*) – The logger to which the regularizer will log its parameters.
- **val_split** (Optional[float]) – The fraction of the training data to use as validation data for the lambda updater. Can be none if no lambda updater is provided.

**Raises**

- **ValueError** – if no lambda updater (lambda_updater) is provided and the initial regularization strength (initial_lambda) is zero.
- **ValueError** – if a validation split (val_split) is provided but it's not a float in the (0, 1) interval.
- **ValueError** – if a lambda updater is provided but no validation split is provided.
- **ValueError** – if a validation split is set, but no lambda updater is provided.

**classmethod create** (*initial_lambda*, *lambda_updater=None*, *val_split=0.0*, ***kwargs*)

Create a regularizer.

**Return type**

*RegularizerFactory*[TypeVar(Self, bound= Regularizer)]

**lambda_: float**

**lambda_updater: Optional[*LambdaUpdater*]**

**logger: *HierarchicalLogger***

**optimizer: Optimizer**

**abstract regularize_and_backward** (*loss*)

Abstract method for performing the regularization step.

The return type is a generic and the specific implementation must describe the meaning of the return type.

This step will also call *loss.backward()* for the user. This is because the regularizer may require the loss to be called before or after the regularization step. Leaving this to the user would force them to make their implementation dependent on the regularizer algorithm used, which is prone to errors.

**Parameters**

**loss** (Tensor) – The loss to regularize.

**Return type**

TypeVar(R)

**update_params** (*train_loss*, *val_loss*)

Update the regularization parameter.

This method calls the lambda_updater to update the regularization parameter, and assigns the new value to *self.lambda_*. Then logs the new value using the provided logger.

**Parameters**

- **train_loss** (Union[Tensor, float]) – The loss on the training set.
- **val_loss** (Union[Tensor, float]) – The loss on the validation set.

**Return type**

None

**val_split: Optional[float]**

**class** imitation.regularization.regularizers.**RegularizerFactory**(**args*, ***kwargs*)

Bases: Protocol[T_Regularizer_co]

Protocol for functions that create regularizers.

The regularizer factory is meant to be used as a way to create a regularizer in two steps. First, the end-user creates a regularizer factory by calling the *.create()* method of a regularizer class. This allows specifying all the relevant configuration to the regularization algorithm. Then, the network algorithm finishes setting up the optimizer and logger, and calls the regularizer factory to create the regularizer.

This two-step process separates the configuration of the regularization algorithm from additional "operational" parameters. This is useful because it solves two problems:

1. The end-user does not have access to the optimizer and logger when configuring the regularization algorithm.
2. Validation of the configuration is done outside the network constructor.

It also allows re-using the same regularizer factory for multiple networks.

**__init__**(**args*, ***kwargs*)

**class** imitation.regularization.regularizers.**WeightDecayRegularizer**(*optimizer*, *initial_lambda*, *lambda_updater*, *logger*, *val_split=None*)

Bases: *WeightRegularizer*

Applies weight decay to a loss function.

**lambda_: float**

**lambda_updater: Optional[*LambdaUpdater*]**

**logger: *HierarchicalLogger***

**optimizer: Optimizer**

**val_split: Optional[float]**

**class** imitation.regularization.regularizers.**WeightRegularizer**(*optimizer*, *initial_lambda*, *lambda_updater*, *logger*, *val_split=None*)

Bases: *Regularizer*

Abstract base class for regularizers that regularize the weights of a network.

Requires the user to implement the _weight_penalty method.

**lambda_: float**

**lambda_updater: Optional[*LambdaUpdater*]**

**logger: *HierarchicalLogger***

**optimizer: Optimizer**

**regularize_and_backward**(*loss*)

Regularize the weights of the network, and call `loss.backward()`.

**Return type**

`None`

**val_split: Optional[float]**

### imitation.regularization.updaters

Implements parameter scaling algorithms to update the parameters of a regularizer.

#### Classes

| | |
|---|---|
| *IntervalParamScaler*(scaling_factor, ...) | Scales the lambda of the regularizer by some constant factor. |
| *LambdaUpdater*(*args, **kwargs) | Protocol type for functions that update the regularizer parameter. |

**class** imitation.regularization.updaters.**IntervalParamScaler**(*scaling_factor*, *tolerable_interval*)

Bases: *LambdaUpdater*

Scales the lambda of the regularizer by some constant factor.

Lambda is scaled up if the ratio of the validation loss to the training loss is above the tolerable interval, and scaled down if the ratio is below the tolerable interval. Nothing happens if the ratio is within the tolerable interval.

**__init__**(*scaling_factor*, *tolerable_interval*)

Initialize the interval parameter scaler.

**Parameters**

- **scaling_factor** (`float`) – The factor by which to scale the lambda, a value in (0, 1).
- **tolerable_interval** (`Tuple[float, float]`) – The interval within which the ratio of the validation loss to the training loss is considered acceptable. A tuple whose first element is at least 0 and the second element is greater than the first.

**Raises**

- **ValueError** – If the tolerable interval is not a tuple of length 2.
- **ValueError** – if the scaling factor is not in (0, 1).
- **ValueError** – if the tolerable interval is negative or not a proper interval.

**class** imitation.regularization.updaters.**LambdaUpdater**(**args*, ***kwargs*)

Bases: `Protocol`

Protocol type for functions that update the regularizer parameter.

A callable object that takes in the current lambda and the train and val loss, and returns the new lambda. This has been implemented as a protocol and not an ABC because a user might wish to provide their own implementation without having to inherit from the base class, e.g. by defining a function instead of a class.

Note: if you implement *LambdaUpdater*, your implementation MUST be purely functional, i.e. side-effect free. The class structure should only be used to store constant hyperparameters. (Alternatively, closures can be used for that).

**__init__**(**args*, ***kwargs*)

## 3.1.5 imitation.rewards

Reward models: neural network modules, serialization, preprocessing, etc.

### Modules

| | |
|---|---|
| *imitation.rewards.reward_function* | Type alias shared by reward-related code. |
| *imitation.rewards.reward_nets* | Constructs deep network reward models. |
| *imitation.rewards.reward_wrapper* | Common wrapper for adding custom reward values to an environment. |
| *imitation.rewards.serialize* | Load serialized reward functions of different types. |

### imitation.rewards.reward_function

Type alias shared by reward-related code.

### Classes

| | |
|---|---|
| *RewardFn*(*args, **kwargs) | Abstract class for reward function. |

**class** imitation.rewards.reward_function.**RewardFn**(**args*, ***kwargs*)

Bases: Protocol

Abstract class for reward function.

Requires implementation of __call__() to compute the reward given a batch of states, actions, next states and dones.

**__init__**(**args*, ***kwargs*)

### imitation.rewards.reward_nets

Constructs deep network reward models.

### Functions

| | |
|---|---|
| *cnn_transpose*(tens) | Transpose a (b,h,w,c)-formatted tensor to (b,c,h,w) format. |

### Classes

| | |
|---|---|
| *AddSTDRewardWrapper*(base[, default_alpha]) | Adds a multiple of the estimated standard deviation to mean reward. |
| *BasicPotentialCNN*(observation_space, hid_sizes) | Simple implementation of a potential using a CNN. |
| *BasicPotentialMLP*(observation_space, ...) | Simple implementation of a potential using an MLP. |
| *BasicRewardNet*(observation_space, action_space) | MLP that takes as input the state, action, next state and done flag. |
| *BasicShapedRewardNet*(observation_space, ...) | Shaped reward net based on MLPs. |
| *CnnRewardNet*(observation_space, action_space) | CNN that takes as input the state, action, next state and done flag. |
| *ForwardWrapper*(base) | An abstract RewardNetWrapper that changes the behavior of forward. |
| *NormalizedRewardNet*(base, normalize_output_layer) | A reward net that normalizes the output of its base network. |
| *PredictProcessedWrapper*(base) | An abstract RewardNetWrapper that changes the behavior of predict_processed. |
| *RewardEnsemble*(observation_space, ...) | A mean ensemble of reward networks. |
| *RewardNet*(observation_space, action_space[, ...]) | Minimal abstract reward network. |
| *RewardNetWithVariance*(observation_space, ...) | A reward net that keeps track of its epistemic uncertainty through variance. |
| *RewardNetWrapper*(base) | Abstract class representing a wrapper modifying a `RewardNet`'s functionality. |
| *ShapedRewardNet*(base, potential, discount_factor) | A RewardNet consisting of a base network and a potential shaping. |

**class** imitation.rewards.reward_nets.**AddSTDRewardWrapper**(*base*, *default_alpha=0.0*)

Bases: *PredictProcessedWrapper*

Adds a multiple of the estimated standard deviation to mean reward.

**__init__**(*base*, *default_alpha=0.0*)

Create a reward network that adds a multiple of the standard deviation.

**Parameters**

- **base** (*RewardNetWithVariance*) – A reward network that keeps track of its epistemic variance. This is used to compute the standard deviation.
- **default_alpha** (`float`) – multiple of standard deviation to add to the reward mean. Defaults to 0.0.

**Raises**

**TypeError** – if base is not an instance of RewardNetWithVariance

**predict_processed**(*state*, *action*, *next_state*, *done*, *alpha=None*, ***kwargs*)

Compute a lower/upper confidence bound on the reward without gradients.

**Parameters**

- **state** (`ndarray`) – Current states of shape *(batch_size,)* + *state_shape*.
- **action** (`ndarray`) – Actions of shape *(batch_size,)* + *action_shape*.
- **next_state** (`ndarray`) – Successor states of shape *(batch_size,)* + *state_shape*.
- **done** (`ndarray`) – End-of-episode (terminal state) indicator of shape *(batch_size,)*.

- **alpha** (Optional[float]) – multiple of standard deviation to add to the reward mean. Defaults to the value provided at initialization.
- ****kwargs** – are not used

**Return type**
ndarray

**Returns**
Estimated lower confidence bounds on rewards of shape *(batch_size,)*.

**class** imitation.rewards.reward_nets.**BasicPotentialCNN**(*observation_space*, *hid_sizes*, *hwc_format=True*, ***kwargs*)

Bases: Module

Simple implementation of a potential using a CNN.

**__init__**(*observation_space*, *hid_sizes*, *hwc_format=True*, ***kwargs*)

Initialize the potential.

**Parameters**

- **observation_space** (Space) – observation space of the environment.
- **hid_sizes** (Iterable[int]) – number of channels in hidden layers of the CNN.
- **hwc_format** (bool) – format of the observation. True if channel dimension is last, False if channel dimension is first.
- **kwargs** – passed straight through to *build_cnn*.

**Raises**
**ValueError** – if observations are not images.

**forward**(*state*)

Define 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.

**Return type**
Tensor

**training: bool**

**class** imitation.rewards.reward_nets.**BasicPotentialMLP**(*observation_space*, *hid_sizes*, ***kwargs*)

Bases: Module

Simple implementation of a potential using an MLP.

**__init__**(*observation_space*, *hid_sizes*, ***kwargs*)

Initialize the potential.

**Parameters**

- **observation_space** (Space) – observation space of the environment.

- **hid_sizes** (Iterable[int]) – widths of the hidden layers of the MLP.
- **kwargs** – passed straight through to *build_mlp*.

**forward** (*state*)

Define 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.

**Return type**

Tensor

**training: bool**

**class** imitation.rewards.reward_nets.**BasicRewardNet** (*observation_space*, *action_space*, *use_state=True*, *use_action=True*, *use_next_state=False*, *use_done=False*, ***kwargs*)

Bases: *RewardNet*

MLP that takes as input the state, action, next state and done flag.

These inputs are flattened and then concatenated to one another. Each input can enabled or disabled by the *use_** constructor keyword arguments.

**__init__** (*observation_space*, *action_space*, *use_state=True*, *use_action=True*, *use_next_state=False*, *use_done=False*, ***kwargs*)

Builds reward MLP.

**Parameters**

- **observation_space** (Space) – The observation space.
- **action_space** (Space) – The action space.
- **use_state** (bool) – should the current state be included as an input to the MLP?
- **use_action** (bool) – should the current action be included as an input to the MLP?
- **use_next_state** (bool) – should the next state be included as an input to the MLP?
- **use_done** (bool) – should the “done” flag be included as an input to the MLP?
- **kwargs** – passed straight through to *build_mlp*.

**forward** (*state*, *action*, *next_state*, *done*)

Compute rewards for a batch of transitions and keep gradients.

**training: bool**

**class** imitation.rewards.reward_nets.**BasicShapedRewardNet**(*observation_space*, *action_space*, ***, *reward_hid_sizes=(32,)*, *potential_hid_sizes=(32, 32)*, *use_state=True*, *use_action=True*, *use_next_state=False*, *use_done=False*, *discount_factor=0.99*, ***kwargs*)

Bases: *ShapedRewardNet*

Shaped reward net based on MLPs.

This is just a very simple convenience class for instantiating a BasicRewardNet and a BasicPotentialMLP and wrapping them inside a ShapedRewardNet. Mainly exists for backwards compatibility after https://github.com/HumanCompatibleAI/imitation/pull/311 to keep the scripts working.

**TODO(ejnnr): if we ever modify AIRL so that it takes in a RewardNet instance**
directly (instead of a class and kwargs) and instead instantiate the RewardNet inside the scripts, then it probably makes sense to get rid of this class.

**__init__**(*observation_space*, *action_space*, ***, *reward_hid_sizes=(32,)*, *potential_hid_sizes=(32, 32)*, *use_state=True*, *use_action=True*, *use_next_state=False*, *use_done=False*, *discount_factor=0.99*, ***kwargs*)

Builds a simple shaped reward network.

**Parameters**

- **observation_space** (Space) – The observation space.
- **action_space** (Space) – The action space.
- **reward_hid_sizes** (Sequence[int]) – sequence of widths for the hidden layers of the base reward MLP.
- **potential_hid_sizes** (Sequence[int]) – sequence of widths for the hidden layers of the potential MLP.
- **use_state** (bool) – should the current state be included as an input to the reward MLP?
- **use_action** (bool) – should the current action be included as an input to the reward MLP?
- **use_next_state** (bool) – should the next state be included as an input to the reward MLP?
- **use_done** (bool) – should the “done” flag be included as an input to the reward MLP?
- **discount_factor** (float) – discount factor for the potential shaping.
- **kwargs** – passed straight through to *BasicRewardNet* and *BasicPotentialMLP*.

**training: bool**

**class** imitation.rewards.reward_nets.**CnnRewardNet**(*observation_space*, *action_space*, *use_state=True*, *use_action=True*, *use_next_state=False*, *use_done=False*, *hwc_format=True*, ***kwargs*)

Bases: *RewardNet*

CNN that takes as input the state, action, next state and done flag.

Inputs are boosted to tensors with channel, height, and width dimensions, and then concatenated. Image inputs are assumed to be in (h,w,c) format, unless the argument hwc_format=False is passed in. Each input can be enabled or disabled by the *use_** constructor keyword arguments, but either *use_state* or *use_next_state* must be True.

**`__init__`**(*observation_space*, *action_space*, *use_state=True*, *use_action=True*, *use_next_state=False*, *use_done=False*, *hwc_format=True*, ***kwargs*)

Builds reward CNN.

**Parameters**

- **`observation_space`** (`Space`) – The observation space.
- **`action_space`** (`Space`) – The action space.
- **`use_state`** (`bool`) – Should the current state be included as an input to the CNN?
- **`use_action`** (`bool`) – Should the current action be included as an input to the CNN?
- **`use_next_state`** (`bool`) – Should the next state be included as an input to the CNN?
- **`use_done`** (`bool`) – Should the "done" flag be included as an input to the CNN?
- **`hwc_format`** (`bool`) – Are image inputs in (h,w,c) format (True), or (c,h,w) (False)? If hwc_format is False, image inputs are not transposed.
- **`kwargs`** – Passed straight through to *build_cnn*.

**Raises**

**`ValueError`** – if observation or action space is not easily massaged into a CNN input.

**`forward`**(*state*, *action*, *next_state*, *done*)

Computes rewardNet value on input state, action, next_state, and done flag.

Takes inputs that will be used, transposes image states to (c,h,w) format if needed, reshapes inputs to have compatible dimensions, concatenates them, and inputs them into the CNN.

**Parameters**

- **`state`** (`Tensor`) – current state.
- **`action`** (`Tensor`) – current action.
- **`next_state`** (`Tensor`) – next state.
- **`done`** (`Tensor`) – flag for whether the episode is over.

**Returns**

reward of the transition.

**Return type**

th.Tensor

**`get_num_channels_obs`**(*space*)

Gets number of channels for the observation.

**Return type**

`int`

**`training: bool`**

**`class`** `imitation.rewards.reward_nets.`**`ForwardWrapper`**(*base*)

Bases: *`RewardNetWrapper`*

An abstract RewardNetWrapper that changes the behavior of forward.

Note that all forward wrappers must be placed before all predict processed wrappers.

**`__init__`**(*base*)

Create a forward wrapper.

**Parameters**

**`base`** (*`RewardNet`*) – The base reward network

**Raises**

**`ValueError`** – if the base network is a *PredictProcessedWrapper*.

**`training: bool`**

`class imitation.rewards.reward_nets.`**`NormalizedRewardNet`**(*base*, *normalize_output_layer*)

Bases: *`PredictProcessedWrapper`*

A reward net that normalizes the output of its base network.

**`__init__`**(*base*, *normalize_output_layer*)

Initialize the NormalizedRewardNet.

**Parameters**

- **`base`** (*`RewardNet`*) – a base RewardNet
- **`normalize_output_layer`** (`Type[`*`BaseNorm`*`]`) – The class to use to normalize rewards. This can be any nn.Module that preserves the shape; e.g. *nn.Identity*, *nn.LayerNorm*, or *networks.RunningNorm*.

**`predict_processed`**(*state*, *action*, *next_state*, *done*, *update_stats=True*, ***kwargs*)

Compute normalized rewards for a batch of transitions without gradients.

**Parameters**

- **`state`** (`ndarray`) – Current states of shape *(batch_size,)* + *state_shape*.
- **`action`** (`ndarray`) – Actions of shape *(batch_size,)* + *action_shape*.
- **`next_state`** (`ndarray`) – Successor states of shape *(batch_size,)* + *state_shape*.
- **`done`** (`ndarray`) – End-of-episode (terminal state) indicator of shape *(batch_size,)*.
- **`update_stats`** (`bool`) – Whether to update the running stats of the normalization layer.
- **`**kwargs`** – kwargs passed to base predict_processed call.

**Return type**

`ndarray`

**Returns**

Computed normalized rewards of shape *(batch_size,)*.

**`training: bool`**

`class imitation.rewards.reward_nets.`**`PredictProcessedWrapper`**(*base*)

Bases: *`RewardNetWrapper`*

An abstract RewardNetWrapper that changes the behavior of predict_processed.

Subclasses should override *predict_processed*. Implementations should pass along *kwargs* to the *base* reward net's *predict_processed* method.

**Note: The wrapper will default to forwarding calls to *device*, *forward*,**
*preprocess* and *predict* to the base reward net unless explicitly overridden in a subclass.

**forward**(*state*, *action*, *next_state*, *done*)

Compute rewards for a batch of transitions and keep gradients.

**Return type**

`Tensor`

**predict**(*state*, *action*, *next_state*, *done*)

Compute rewards for a batch of transitions without gradients.

Converting th.Tensor rewards from *predict_th* to NumPy arrays.

**Parameters**

- **state** (`ndarray`) – Current states of shape *(batch_size,)* + *state_shape*.
- **action** (`ndarray`) – Actions of shape *(batch_size,)* + *action_shape*.
- **next_state** (`ndarray`) – Successor states of shape *(batch_size,)* + *state_shape*.
- **done** (`ndarray`) – End-of-episode (terminal state) indicator of shape *(batch_size,)*.

**Return type**

`ndarray`

**Returns**

Computed rewards of shape *(batch_size,)*.

**abstract predict_processed**(*state*, *action*, *next_state*, *done*, ***kwargs*)

Predict processed must be overridden in subclasses.

**Return type**

`ndarray`

**predict_th**(*state*, *action*, *next_state*, *done*)

Compute th.Tensor rewards for a batch of transitions without gradients.

Preprocesses the inputs, output th.Tensor reward arrays.

**Parameters**

- **state** (`ndarray`) – Current states of shape *(batch_size,)* + *state_shape*.
- **action** (`ndarray`) – Actions of shape *(batch_size,)* + *action_shape*.
- **next_state** (`ndarray`) – Successor states of shape *(batch_size,)* + *state_shape*.
- **done** (`ndarray`) – End-of-episode (terminal state) indicator of shape *(batch_size,)*.

**Return type**

`Tensor`

**Returns**

Computed th.Tensor rewards of shape *(batch_size,*).

**training: bool**

**class** imitation.rewards.reward_nets.**RewardEnsemble**(*observation_space*, *action_space*, *members*)

Bases: *RewardNetWithVariance*

A mean ensemble of reward networks.

A reward ensemble is made up of individual reward networks. To maintain consistency the "output" of a reward network will be defined as the results of its *predict_processed*. Thus for example the mean of the ensemble is the mean of the results of its members predict processed classes.

**`__init__`**(*observation_space*, *action_space*, *members*)

Initialize the RewardEnsemble.

**Parameters**

- **observation_space** (`Space`) – the observation space of the environment
- **action_space** (`Space`) – the action space of the environment
- **members** (`Iterable[RewardNet]`) – the member networks that will make up the ensemble.

**Raises**

**ValueError** – if num_members is less than 1

**`forward`**(**args*)

The forward method of the ensemble should in general not be used directly.

**Return type**

`Tensor`

**`members: ModuleList`**

**`property num_members`**

The number of members in the ensemble.

**`predict`**(*state*, *action*, *next_state*, *done*, ***kwargs*)

Return the mean of the ensemble members.

**`predict_processed`**(*state*, *action*, *next_state*, *done*, ***kwargs*)

Return the mean of the ensemble members.

**Return type**

`ndarray`

**`predict_processed_all`**(*state*, *action*, *next_state*, *done*, ***kwargs*)

Get the results of predict processed on all of the members.

**Parameters**

- **state** (`ndarray`) – Current states of shape *(batch_size,)* + *state_shape*.
- **action** (`ndarray`) – Actions of shape *(batch_size,)* + *action_shape*.
- **next_state** (`ndarray`) – Successor states of shape *(batch_size,)* + *state_shape*.
- **done** (`ndarray`) – End-of-episode (terminal state) indicator of shape *(batch_size,)*.
- **kwargs** – passed along to ensemble members.

**Return type**

`ndarray`

**Returns**

**The result of predict processed for each member in the ensemble of**
shape *(batch_size, num_members)*.

**`predict_reward_moments`**(*state*, *action*, *next_state*, *done*, ***kwargs*)

Compute the standard deviation of the reward distribution for a batch.

**Parameters**

- **state** (`ndarray`) – Current states of shape *(batch_size,)* + *state_shape*.

- **action** (ndarray) – Actions of shape *(batch_size,)* + *action_shape*.
- **next_state** (ndarray) – Successor states of shape *(batch_size,)* + *state_shape*.
- **done** (ndarray) – End-of-episode (terminal state) indicator of shape *(batch_size,)*.
- ****kwargs** – passed along to predict processed.

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

**Returns**

- Reward mean of shape *(batch_size,)*.
- Reward variance of shape *(batch_size,)*.

**class** imitation.rewards.reward_nets.**RewardNet**(*observation_space*, *action_space*, *normalize_images=True*)

Bases: Module, ABC

Minimal abstract reward network.

Only requires the implementation of a forward pass (calculating rewards given a batch of states, actions, next states and dones).

**__init__**(*observation_space*, *action_space*, *normalize_images=True*)

Initialize the RewardNet.

**Parameters**

- **observation_space** (Space) – the observation space of the environment
- **action_space** (Space) – the action space of the environment
- **normalize_images** (bool) – whether to automatically normalize image observations to [0, 1] (from 0 to 255). Defaults to True.

**property device: device**

Heuristic to determine which device this module is on.

**Return type**
device

**property dtype: dtype**

Heuristic to determine dtype of module.

**Return type**
dtype

**abstract forward**(*state*, *action*, *next_state*, *done*)

Compute rewards for a batch of transitions and keep gradients.

**Return type**
Tensor

**predict**(*state*, *action*, *next_state*, *done*)

Compute rewards for a batch of transitions without gradients.

Converting th.Tensor rewards from *predict_th* to NumPy arrays.

**Parameters**

- **state** (ndarray) – Current states of shape *(batch_size,)* + *state_shape*.

- **action** (ndarray) – Actions of shape *(batch_size,)* + *action_shape*.
- **next_state** (ndarray) – Successor states of shape *(batch_size,)* + *state_shape*.
- **done** (ndarray) – End-of-episode (terminal state) indicator of shape *(batch_size,)*.

**Return type**
ndarray

**Returns**
Computed rewards of shape *(batch_size,)*.

**predict_processed** (*state*, *action*, *next_state*, *done*, ***kwargs*)

Compute the processed rewards for a batch of transitions without gradients.

Defaults to calling *predict*. Subclasses can override this to normalize or otherwise modify the rewards in ways that may help RL training or other applications of the reward function.

**Parameters**

- **state** (ndarray) – Current states of shape *(batch_size,)* + *state_shape*.
- **action** (ndarray) – Actions of shape *(batch_size,)* + *action_shape*.
- **next_state** (ndarray) – Successor states of shape *(batch_size,)* + *state_shape*.
- **done** (ndarray) – End-of-episode (terminal state) indicator of shape *(batch_size,)*.
- **kwargs** – additional kwargs may be passed to change the functionality of subclasses.

**Return type**
ndarray

**Returns**
Computed processed rewards of shape *(batch_size,)*.

**predict_th** (*state*, *action*, *next_state*, *done*)

Compute th.Tensor rewards for a batch of transitions without gradients.

Preprocesses the inputs, output th.Tensor reward arrays.

**Parameters**

- **state** (ndarray) – Current states of shape *(batch_size,)* + *state_shape*.
- **action** (ndarray) – Actions of shape *(batch_size,)* + *action_shape*.
- **next_state** (ndarray) – Successor states of shape *(batch_size,)* + *state_shape*.
- **done** (ndarray) – End-of-episode (terminal state) indicator of shape *(batch_size,)*.

**Return type**
Tensor

**Returns**
Computed th.Tensor rewards of shape *(batch_size,)*.

**preprocess** (*state*, *action*, *next_state*, *done*)

Preprocess a batch of input transitions and convert it to PyTorch tensors.

The output of this function is suitable for its forward pass, so a typical usage would be `model(*model.preprocess(transitions))`.

**Parameters**

- **state** (ndarray) – The observation input. Its shape is *(batch_size,)* + *observation_space.shape*.

- **action** (ndarray) – The action input. Its shape is *(batch_size,)* + *action_space.shape*. The None dimension is expected to be the same as None dimension from *obs_input*.
- **next_state** (ndarray) – The observation input. Its shape is *(batch_size,)* + *observation_space.shape*.
- **done** (ndarray) – Whether the episode has terminated. Its shape is *(batch_size,)*.

**Returns**
a Tuple of tensors containing observations, actions, next observations and dones.

**Return type**
Preprocessed transitions

**training: bool**

**class** imitation.rewards.reward_nets.**RewardNetWithVariance**(*observation_space*, *action_space*, *normalize_images=True*)

Bases: *RewardNet*

A reward net that keeps track of its epistemic uncertainty through variance.

**abstract predict_reward_moments**(*state*, *action*, *next_state*, *done*, ***kwargs*)

Compute the mean and variance of the reward distribution.

**Parameters**

- **state** (ndarray) – Current states of shape *(batch_size,)* + *state_shape*.
- **action** (ndarray) – Actions of shape *(batch_size,)* + *action_shape*.
- **next_state** (ndarray) – Successor states of shape *(batch_size,)* + *state_shape*.
- **done** (ndarray) – End-of-episode (terminal state) indicator of shape *(batch_size,)*.
- ****kwargs** – may modify the behavior of subclasses

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

**Returns**

- Estimated reward mean of shape *(batch_size,)*.
- Estimated reward variance of shape *(batch_size,)*. # noqa: DAR202

**training: bool**

**class** imitation.rewards.reward_nets.**RewardNetWrapper**(*base*)

Bases: *RewardNet*

Abstract class representing a wrapper modifying a RewardNet's functionality.

In general RewardNetWrapper``s should either subclass ``ForwardWrapper or PredictProcessedWrapper.

**__init__**(*base*)

Initialize a RewardNet wrapper.

**Parameters**
**base** (*RewardNet*) – the base RewardNet to wrap.

**property base: *RewardNet***

**Return type**
*RewardNet*

**property device: device**

Heuristic to determine which device this module is on.

**Return type**
device

**property dtype: dtype**

Heuristic to determine dtype of module.

**Return type**
dtype

**preprocess**(*state*, *action*, *next_state*, *done*)

Preprocess a batch of input transitions and convert it to PyTorch tensors.

The output of this function is suitable for its forward pass, so a typical usage would be `model(*model.preprocess(transitions))`.

**Parameters**

- **state** (ndarray) – The observation input. Its shape is *(batch_size,)* + *observation_space.shape*.
- **action** (ndarray) – The action input. Its shape is *(batch_size,)* + *action_space.shape*. The None dimension is expected to be the same as None dimension from *obs_input*.
- **next_state** (ndarray) – The observation input. Its shape is *(batch_size,)* + *observation_space.shape*.
- **done** (ndarray) – Whether the episode has terminated. Its shape is *(batch_size,)*.

**Returns**
a Tuple of tensors containing observations, actions, next observations and dones.

**Return type**
Preprocessed transitions

**training: bool**

**class** imitation.rewards.reward_nets.**ShapedRewardNet**(*base*, *potential*, *discount_factor*)

Bases: *ForwardWrapper*

A RewardNet consisting of a base network and a potential shaping.

**__init__**(*base*, *potential*, *discount_factor*)

Setup a ShapedRewardNet instance.

**Parameters**

- **base** (*RewardNet*) – the base reward net to which the potential shaping will be added. Shaping must be applied directly to the raw reward net. See error below.
- **potential** (Callable[[Tensor], Tensor]) – A callable which takes a batch of states (as a PyTorch tensor) and returns a batch of potentials for these states. If this is a PyTorch Module, it becomes a submodule of the ShapedRewardNet instance.
- **discount_factor** (float) – discount factor to use for the potential shaping.

**forward**(*state*, *action*, *next_state*, *done*)

Compute rewards for a batch of transitions and keep gradients.

**training: bool**

imitation.rewards.reward_nets.**cnn_transpose**(*tens*)

Transpose a (b,h,w,c)-formatted tensor to (b,c,h,w) format.

**Return type**

Tensor

### imitation.rewards.reward_wrapper

Common wrapper for adding custom reward values to an environment.

### Classes

| | |
|---|---|
| *RewardVecEnvWrapper*(venv, reward_fn[, ...]) | Uses a provided reward_fn to replace the reward function returned by *step()*. |
| *WrappedRewardCallback*(episode_rewards, ...) | Logs mean wrapped reward as part of RL (or other) training. |

**class** imitation.rewards.reward_wrapper.**RewardVecEnvWrapper**(*venv*, *reward_fn*, *ep_history=100*)

Bases: VecEnvWrapper

Uses a provided reward_fn to replace the reward function returned by *step()*.

Automatically resets the inner VecEnv upon initialization. A tricky part about this class is keeping track of the most recent observation from each environment.

Will also include the previous reward given by the inner VecEnv in the returned info dict under the *original_env_rew* key.

**__init__**(*venv*, *reward_fn*, *ep_history=100*)

Builds RewardVecEnvWrapper.

**Parameters**

- **venv** (VecEnv) – The VecEnv to wrap.
- **reward_fn** (*RewardFn*) – A function that wraps takes in vectorized transitions (obs, act, next_obs) a vector of episode timesteps, and returns a vector of rewards.
- **ep_history** (int) – The number of episode rewards to retain for computing mean reward.

**property envs**

**make_log_callback**()

Creates *WrappedRewardCallback* connected to this *RewardVecEnvWrapper*.

**Return type**

*WrappedRewardCallback*

**reset**()

Reset all the environments and return an array of observations, or a tuple of observation arrays.

If step_async is still doing work, that work will be cancelled and step_wait() should not be called until step_async() is invoked again.

**Returns**

observation

**step_async**(*actions*)

Tell all the environments to start taking a step with the given actions. Call step_wait() to get the results of the step.

You should not call this if a step_async run is already pending.

**step_wait**()

Wait for the step taken with step_async().

**Returns**

observation, reward, done, information

**class** `imitation.rewards.reward_wrapper.`**WrappedRewardCallback**(*episode_rewards*, **args*, ***kwargs*)

Bases: `BaseCallback`

Logs mean wrapped reward as part of RL (or other) training.

**__init__**(*episode_rewards*, **args*, ***kwargs*)

Builds WrappedRewardCallback.

**Parameters**

- **episode_rewards** (`Deque[float]`) – A queue that episode rewards will be placed into.
- ***args** – Passed through to *callbacks.BaseCallback*.
- ****kwargs** – Passed through to *callbacks.BaseCallback*.

**model: base_class.BaseAlgorithm**

## imitation.rewards.serialize

Load serialized reward functions of different types.

### Functions

| | |
|---|---|
| *load_reward*(reward_type, reward_path, venv, ...) | Load serialized reward. |
| *load_zero*(path, venv) | **rtype**<br>*RewardFn* |

### Classes

| | |
|---|---|
| *ValidateRewardFn*(reward_fn) | Wrap reward function to add sanity check. |

**class** imitation.rewards.serialize.**ValidateRewardFn**(*reward_fn*)

Bases: *RewardFn*

Wrap reward function to add sanity check.

Checks that the length of the reward vector is equal to the batch size of the input.

**__init__**(*reward_fn*)

Builds the reward validator.

**Parameters**

**reward_fn** (*RewardFn*) – base reward function

imitation.rewards.serialize.**load_reward**(*reward_type*, *reward_path*, *venv*, ***kwargs*)

Load serialized reward.

**Parameters**

- **reward_type** (str) – A key in *reward_registry*. Valid types include RewardNet_normalized, RewardNet_unshaped, zero, RewardNet_unnormalized, RewardNet_shaped, RewardNet_std_added.
- **reward_path** (str) – A path specifying the reward.
- **venv** (VecEnv) – An environment that the policy is to be used with.
- ****kwargs** – kwargs to pass to reward fn

**Return type**

*RewardFn*

**Returns**

The deserialized reward.

imitation.rewards.serialize.**load_zero**(*path*, *venv*)

**Return type**

*RewardFn*

## 3.1.6 imitation.scripts

Command-line scripts.

### Modules

| | |
|---|---|
| *imitation.scripts.analyze* | Commands to analyze experimental results. |
| *imitation.scripts.config* | Configuration settings for scripts. |
| *imitation.scripts.convert_trajs* | Converts old-style pickle or npz trajectories to new-style HuggingFace datasets. |
| *imitation.scripts.eval_policy* | Evaluate policies: render policy interactively, save videos, log episode return. |
| *imitation.scripts.ingredients* | Ingredients for Sacred experiments. |
| *imitation.scripts.parallel* | Runs a Sacred experiment in parallel. |
| *imitation.scripts.train_adversarial* | Train GAIL or AIRL. |
| *imitation.scripts.train_imitation* | Trains DAgger on synthetic demonstrations generated from an expert policy. |
| *imitation.scripts.train_preference_comparisons* | Train a reward model using preference comparisons. |
| *imitation.scripts.train_rl* | Uses RL to train a policy from scratch, saving rollouts and policy. |
| *imitation.scripts.tuning* | Tunes the hyperparameters of the algorithms. |

### imitation.scripts.analyze

Commands to analyze experimental results.

### Functions

| | |
|---|---|
| *analyze_imitation*(csv_output_path, ...) | Parse Sacred logs and generate a DataFrame for imitation learning results. |
| *gather_tb_directories*() | Gather Tensorboard directories from a *parallel_ex* run. |
| *main_console*() | |

imitation.scripts.analyze.**analyze_imitation**(*csv_output_path*, *tex_output_path*, *print_table*, *table_verbosity*)

Parse Sacred logs and generate a DataFrame for imitation learning results.

This function calls the helper *_gather_sacred_dicts*, which captures its arguments automatically via Sacred. Provide those arguments to select which Sacred results to parse.

**Parameters**

- **csv_output_path** (Optional[str]) – If provided, then save a CSV output file to this path.
- **tex_output_path** (Optional[str]) – If provided, then save a LaTeX-format table to this path.
- **print_table** (bool) – If True, then print the dataframe to stdout.
- **table_verbosity** (int) – Increasing levels of verbosity, from 0 to 3, increase the number of columns in the table. Level 3 prints all of the columns available.

**Return type**

DataFrame

**Returns**

The DataFrame generated from the Sacred logs.

`imitation.scripts.analyze.`**`gather_tb_directories`**`()`

Gather Tensorboard directories from a *parallel_ex* run.

The directories are copied to a unique directory in */tmp/analysis_tb/* under subdirectories matching the Tensorboard events' Ray Tune trial names.

This function calls the helper *_gather_sacred_dicts*, which captures its arguments automatically via Sacred. Provide those arguments to select which Sacred results to parse.

**Return type**

`dict`

**Returns**

A dict with two keys. "gather_dir" (str) is a path to a /tmp/ directory containing all the TensorBoard runs filtered from *source_dir*. "n_tb_dirs" (int) is the number of TensorBoard directories that were filtered.

**Raises**

**`OSError`** – If the symlink cannot be created.

`imitation.scripts.analyze.`**`main_console`**`()`

### imitation.scripts.config

Configuration settings for scripts.

#### Modules

| | |
|---|---|
| *imitation.scripts.config.analyze* | Configuration settings for analyze, inspecting results from completed experiments. |
| *imitation.scripts.config.eval_policy* | Configuration settings for eval_policy, evaluating pre-trained policies. |
| *imitation.scripts.config.parallel* | Config files for parallel experiments. |
| *imitation.scripts.config.train_adversarial* | Configuration for imitation.scripts.train_adversarial. |
| *imitation.scripts.config.train_imitation* | Configuration settings for train_dagger, training DAgger from synthetic demos. |
| *imitation.scripts.config.train_preference_comparisons* | Configuration for imitation.scripts.train_preference_comparisons. |
| *imitation.scripts.config.train_rl* | Configuration settings for train_rl, training a policy with RL. |
| *imitation.scripts.config.tuning* | Config files for tuning experiments. |

### imitation.scripts.config.analyze

Configuration settings for analyze, inspecting results from completed experiments.

### imitation.scripts.config.eval_policy

Configuration settings for eval_policy, evaluating pre-trained policies.

### imitation.scripts.config.parallel

Config files for parallel experiments.

Parallel experiments are intended to be defined in Python rather than via CLI. For example, a user should add a new *@parallel_ex.named_config* to define a new parallel experiment.

Adding custom named configs is necessary because the CLI interface can't add search spaces to the config like *"seed": tune.choice([0, 1, 2, 3])*.

For tuning hyperparameters of an algorithm on a given environment, check out the imitation/scripts/tuning.py script.

### imitation.scripts.config.train_adversarial

Configuration for imitation.scripts.train_adversarial.

### imitation.scripts.config.train_imitation

Configuration settings for train_dagger, training DAgger from synthetic demos.

### imitation.scripts.config.train_preference_comparisons

Configuration for imitation.scripts.train_preference_comparisons.

### imitation.scripts.config.train_rl

Configuration settings for train_rl, training a policy with RL.

### imitation.scripts.config.tuning

Config files for tuning experiments.

### imitation.scripts.convert_trajs

Converts old-style pickle or npz trajectories to new-style HuggingFace datasets.

See https://github.com/HumanCompatibleAI/imitation/pull/448 for a description of the new trajectory format.

This script takes as command-line input multiple paths to saved trajectories, in the old .pkl or .npz format. It then saves each sequence in the new HuggingFace datasets format. The path is the same as the original but a directory without an extension (i.e. "A.pkl" -> "A/", "A.npz" -> "A/", "A/" -> "A/", "A.foo" -> "A/").

#### Functions

| | |
|---|---|
| *main*() | |
| *update_traj_file_in_place*(path_str, /) | Converts pickle or npz file to the new HuggingFace format. |

`imitation.scripts.convert_trajs.`**`main`**`()`

`imitation.scripts.convert_trajs.`**`update_traj_file_in_place`**`(`*path_str*`, /)`

Converts pickle or npz file to the new HuggingFace format.

The new data is saved as *Sequence[imitation.types.TrajectoryWithRew]*.

**Parameters**

**`path_str`** (`Union[str, bytes, PathLike]`) – Path to a pickle or npz file containing *Sequence[imitation.types.Trajectory]* or *Sequence[imitation.old_types.TrajectoryWithRew]*.

**Return type**

`Path`

**Returns**

The path to the converted trajectory dataset.

### imitation.scripts.eval_policy

Evaluate policies: render policy interactively, save videos, log episode return.

#### Functions

| | |
|---|---|
| *eval_policy*(eval_n_timesteps, ...[, ...]) | Rolls a policy out in an environment, collecting statistics. |
| *main_console*() | |
| *video_wrapper_factory*(log_dir, **kwargs) | Returns a function that wraps the environment in a video recorder. |

### Classes

| | |
|---|---|
| *InteractiveRender*(venv, fps) | Render the wrapped environment(s) on screen. |

**class** imitation.scripts.eval_policy.**InteractiveRender**(*venv*, *fps*)

Bases: VecEnvWrapper

Render the wrapped environment(s) on screen.

**__init__**(*venv*, *fps*)

Builds renderer for *venv* running at *fps* frames per second.

**reset**()

Reset all the environments and return an array of observations, or a tuple of observation arrays.

If step_async is still doing work, that work will be cancelled and step_wait() should not be called until step_async() is invoked again.

**Returns**
observation

**step_wait**()

Wait for the step taken with step_async().

**Returns**
observation, reward, done, information

imitation.scripts.eval_policy.**eval_policy**(*eval_n_timesteps*, *eval_n_episodes*, *render*, *render_fps*, *videos*, *video_kwargs*, *_run*, *_rnd*, *reward_type=None*, *reward_path=None*, *rollout_save_path=None*, *explore_kwargs=None*)

Rolls a policy out in an environment, collecting statistics.

**Parameters**

- **eval_n_timesteps** (Optional[int]) – Minimum number of timesteps to evaluate for. Set exactly one of *eval_n_episodes* and *eval_n_timesteps*.
- **eval_n_episodes** (Optional[int]) – Minimum number of episodes to evaluate for. Set exactly one of *eval_n_episodes* and *eval_n_timesteps*.
- **render** (bool) – If True, renders interactively to the screen.
- **render_fps** (int) – The target number of frames per second to render on screen.
- **videos** (bool) – If True, saves videos to *log_dir*.
- **video_kwargs** (Mapping[str, Any]) – Keyword arguments passed through to *video_wrapper.VideoWrapper*.
- **_rnd** (Generator) – Random number generator provided by Sacred.
- **reward_type** (Optional[str]) – If specified, overrides the environment reward with a reward of this.
- **reward_path** (Optional[str]) – If reward_type is specified, the path to a serialized reward of *reward_type* to override the environment reward with.
- **rollout_save_path** (Optional[str]) – where to save rollouts used for computing stats to disk; if None, then do not save.

- **explore_kwargs** (Optional[Mapping[str, Any]]) – keyword arguments to an exploration wrapper to apply before rolling out, not including policy_callable, venv, and rng; if None, then do not wrap.

**Returns**
Return value of *imitation.util.rollout.rollout_stats()*.

imitation.scripts.eval_policy.**main_console**()

imitation.scripts.eval_policy.**video_wrapper_factory**(*log_dir*, ***kwargs*)

Returns a function that wraps the environment in a video recorder.

### imitation.scripts.ingredients

Ingredients for Sacred experiments.

### Modules

| | |
|---|---|
| *imitation.scripts.ingredients.bc* | This ingredient provides BC algorithm instance. |
| *imitation.scripts.ingredients.demonstrations* | This ingredient provides (expert) demonstrations to learn from. |
| *imitation.scripts.ingredients.environment* | This ingredient provides a vectorized gym environment. |
| *imitation.scripts.ingredients.expert* | This ingredient provides an expert policy. |
| *imitation.scripts.ingredients.logging* | This ingredient provides a number of logging utilities. |
| *imitation.scripts.ingredients.policy* | This ingredient provides a newly constructed stable-baselines3 policy. |
| *imitation.scripts.ingredients.policy_evaluation* | This ingredient performs evaluation of learned policy. |
| *imitation.scripts.ingredients.reward* | This ingredient provides a reward network. |
| *imitation.scripts.ingredients.rl* | This ingredient provides a reinforcement learning algorithm from stable-baselines3. |
| *imitation.scripts.ingredients.sqil* | This ingredient provides a SQIL algorithm instance. |
| *imitation.scripts.ingredients.wb* | This ingredient provides Weights & Biases logging. |

### imitation.scripts.ingredients.bc

This ingredient provides BC algorithm instance.

It is either loaded from disk or constructed from scratch.

### Functions

| | |
|---|---|
| *make_bc*(venv, expert_trajs, custom_logger, ...) | **rtype**<br>*BC* |
| *make_or_load_policy*(venv, agent_path) | Makes a policy or loads a policy from a path if provided. |

imitation.scripts.ingredients.bc.**make_bc**(*venv*, *expert_trajs*, *custom_logger*, *batch_size*, *l2_weight*, *optimizer_cls*, *optimizer_kwargs*, *_rnd*)

**Return type**
*BC*

imitation.scripts.ingredients.bc.**make_or_load_policy**(*venv*, *agent_path*)

Makes a policy or loads a policy from a path if provided.

**Parameters**

- **venv** (VecEnv) – Vectorized environment we will be imitating demos from.
- **agent_path** (Optional[str]) – Path to serialized policy. If provided, then load the policy from this path. Otherwise, make a new policy. Specify only if policy_cls and policy_kwargs are not specified.

**Returns**
A Stable Baselines3 policy.

### imitation.scripts.ingredients.demonstrations

This ingredient provides (expert) demonstrations to learn from.

The demonstrations are either loaded from disk, from the HuggingFace Dataset Hub, or sampled from the expert policy provided by the expert ingredient.

### Functions

| | |
|---|---|
| *get_expert_trajectories*(source, path) | Loads expert demonstrations. |

imitation.scripts.ingredients.demonstrations.**get_expert_trajectories**(*source*, *path*)

Loads expert demonstrations.

**Parameters**

- **source** (str) – Can be either *local* to load rollouts from the disk, *huggingface* to load from the HuggingFace hub or *generated* to generate the expert trajectories.
- **path** (str) – A path containing a pickled sequence of *sources.Trajectory*.

**Return type**
Sequence[*Trajectory*]

**Returns**
The expert trajectories.

**Raises**

`ValueError` – if *source* is not in [“local”, “huggingface”, “generated”].

### imitation.scripts.ingredients.environment

This ingredient provides a vectorized gym environment.

#### Functions

| | |
|---|---|
| *make_rollout_venv*(gym_id, num_vec, parallel, ...) | Builds the vector environment for rollouts. |
| *make_venv*(gym_id, num_vec, parallel, ...) | Builds the vector environment. |

imitation.scripts.ingredients.environment.**make_rollout_venv**(*gym_id*, *num_vec*, *parallel*, *max_episode_steps*, *env_make_kwargs*, *_rnd*)

Builds the vector environment for rollouts.

This environment does no logging, and it is wrapped in a *RolloutInfoWrapper*.

**Parameters**

- **gym_id** (str) – The id of the environment to create.
- **num_vec** (int) – Number of *gym.Env* instances to combine into a vector environment.
- **parallel** (bool) – Whether to use “true” parallelism. If True, then use *SubProcVecEnv*. Otherwise, use *DummyVecEnv* which steps through environments serially.
- **max_episode_steps** (int) – If not None, then a TimeLimit wrapper is applied to each environment to artificially limit the maximum number of timesteps in an episode.
- **env_make_kwargs** (Mapping[str, Any]) – The kwargs passed to *spec.make* of a gym environment.
- **_rnd** (Generator) – Random number generator provided by Sacred.

**Yields**

The constructed vector environment.

**Return type**

Generator[VecEnv, None, None]

imitation.scripts.ingredients.environment.**make_venv**(*gym_id*, *num_vec*, *parallel*, *max_episode_steps*, *env_make_kwargs*, *_run*, *_rnd*, ***kwargs*)

Builds the vector environment.

**Parameters**

- **gym_id** (str) – The id of the environment to create.
- **num_vec** (int) – Number of *gym.Env* instances to combine into a vector environment.
- **parallel** (bool) – Whether to use “true” parallelism. If True, then use *SubProcVecEnv*. Otherwise, use *DummyVecEnv* which steps through environments serially.
- **max_episode_steps** (int) – If not None, then a TimeLimit wrapper is applied to each environment to artificially limit the maximum number of timesteps in an episode.

- **env_make_kwargs** (Mapping[str, Any]) – The kwargs passed to *spec.make* of a gym environment.
- **kwargs** – Passed through to *util.make_vec_env*.

**Yields**
The constructed vector environment.

**Return type**
Generator[VecEnv, None, None]

### imitation.scripts.ingredients.expert

This ingredient provides an expert policy.

The expert policy is either loaded from disk or from the HuggingFace Model Hub or is a test policy (e.g., random or zero). The supported policy types are:

- **ppo and sac: A policy trained with SB3.**
  Needs a *path* in the *loader_kwargs*.
- **<algo>-huggingface (algo can be *ppo* or *sac*):**
  A policy trained with SB3 and uploaded to the HuggingFace Model Hub. Will load the model from the repo <organization>/<algo>-<env_name>. You can set the organization with the *organization* key in loader_kwargs. The default is *HumanCompatibleAI*.
- random: A policy that takes random actions.
- zero: A policy that takes zero actions.

#### Functions

| | |
|---|---|
| *config_hook*(config, command_name, logger) | |
| *get_expert_policy*(venv, policy_type, ...) | |

imitation.scripts.ingredients.expert.**config_hook**(*config*, *command_name*, *logger*)

imitation.scripts.ingredients.expert.**get_expert_policy**(*venv*, *policy_type*, *loader_kwargs*)

### imitation.scripts.ingredients.logging

This ingredient provides a number of logging utilities.

It is responsible for logging to WandB, TensorBoard, and stdout. It will also create a symlink to the sacred logging directory in the log directory.

### Functions

| | |
|---|---|
| *hook*(config, command_name, logger) | |
| *make_log_dir*(_run, log_dir, log_level) | Creates log directory and sets up symlink to Sacred logs. |
| *setup_logging*(_run, log_format_strs) | Builds the imitation logger. |

imitation.scripts.ingredients.logging.**hook**(*config*, *command_name*, *logger*)

imitation.scripts.ingredients.logging.**make_log_dir**(*_run*, *log_dir*, *log_level*)

Creates log directory and sets up symlink to Sacred logs.

**Parameters**

- **log_dir** (str) – The directory to log to.
- **log_level** (Union[int, str]) – The threshold of the logger. Either an integer level (10, 20, …), a string of digits ('10', '20'), or a string of the designated level ('DEBUG', 'INFO', …).

**Return type**

Path

**Returns**

The *log_dir*. This avoids the caller needing to capture this argument.

imitation.scripts.ingredients.logging.**setup_logging**(*_run*, *log_format_strs*)

Builds the imitation logger.

**Parameters**

**log_format_strs** (Sequence[str]) – The types of formats to log to.

**Return type**

Tuple[*HierarchicalLogger*, Path]

**Returns**

The configured imitation logger and *log_dir*. Returning *log_dir* avoids the caller needing to capture this value.

## imitation.scripts.ingredients.policy

This ingredient provides a newly constructed stable-baselines3 policy.

### Functions

| | |
|---|---|
| *make_policy*(venv, policy_cls, policy_kwargs) | Makes policy. |

imitation.scripts.ingredients.policy.**make_policy**(*venv*, *policy_cls*, *policy_kwargs*)

Makes policy.

**Parameters**

- **venv** (VecEnv) – Vectorized environment we will be imitating demos from.
- **policy_cls** (Type[BasePolicy]) – Type of a Stable Baselines3 policy architecture. Specify only if policy_path is not specified.

- **policy_kwargs** (Mapping[str, Any]) – Keyword arguments for policy constructor. Specify only if policy_path is not specified.

**Return type**
BasePolicy

**Returns**
A Stable Baselines3 policy.

### imitation.scripts.ingredients.policy_evaluation

This ingredient performs evaluation of learned policy.

It takes care of the right wrappers, does some rollouts and computes statistics of the rollouts.

#### Functions

| | |
|---|---|
| *eval_policy*(rl_algo, venv, n_episodes_eval, _rnd) | Evaluation of imitation learned policy. |

imitation.scripts.ingredients.policy_evaluation.**eval_policy**(*rl_algo*, *venv*, *n_episodes_eval*, *_rnd*)

Evaluation of imitation learned policy.

Has the side effect of setting *rl_algo*'s environment to *venv* if it is a *BaseAlgorithm*.

**Parameters**

- **rl_algo** (Union[BaseAlgorithm, BasePolicy]) – Algorithm to evaluate.
- **venv** (VecEnv) – Environment to evaluate on.
- **n_episodes_eval** (int) – The number of episodes to average over when calculating the average episode reward of the imitation policy for return.
- **_rnd** (Generator) – Random number generator provided by Sacred.

**Return type**
Mapping[str, float]

**Returns**
A dictionary with two keys. "imit_stats" gives the return value of *rollout_stats()* on rollouts test-reward-wrapped environment, using the final policy (remember that the ground-truth reward can be recovered from the "monitor_return" key). "expert_stats" gives the return value of *rollout_stats()* on the expert demonstrations loaded from *path*.

### imitation.scripts.ingredients.reward

This ingredient provides a reward network.

### Functions

| | |
|---|---|
| *config_hook*(config, command_name, logger) | Sets default values for *net_cls* and *net_kwargs*. |
| *make_reward_net*(venv, net_cls, net_kwargs, ...) | Builds a reward network. |

imitation.scripts.ingredients.reward.**config_hook**(*config*, *command_name*, *logger*)

Sets default values for *net_cls* and *net_kwargs*.

imitation.scripts.ingredients.reward.**make_reward_net**(*venv*, *net_cls*, *net_kwargs*, *normalize_output_layer*, *add_std_alpha*, *ensemble_size*, *ensemble_member_config*)

Builds a reward network.

**Parameters**

- **venv** (VecEnv) – Vectorized environment reward network will predict reward for.
- **net_cls** (Type[*RewardNet*]) – Class of reward network to construct.
- **net_kwargs** (Mapping[str, Any]) – Keyword arguments passed to reward network constructor.
- **normalize_output_layer** (Optional[Type[*BaseNorm*]]) – Wrapping the reward_net with NormalizedRewardNet to normalize the reward output.
- **add_std_alpha** (Optional[float]) – multiple of reward function standard deviation to add to the reward in predict_processed. Must be None when using a reward function that does not keep track of variance. Defaults to None.
- **ensemble_size** (Optional[int]) – The number of ensemble members to create. Must set if using *net_cls* = :class: *reward_nets.RewardEnsemble*.
- **ensemble_member_config** (Optional[Mapping[str, Any]]) – The configuration for individual ensemble members. Note that *ensemble_member_config.net_cls* must not be :class: *reward_nets.RewardEnsemble*. Must be set if using *net_cls* = *` :class: `reward_nets.RewardEnsemble*.

**Return type**

*RewardNet*

**Returns**

A, possibly wrapped, instance of *net_cls*.

**Raises**

**ValueError** – Using a reward ensemble but failed to provide configuration.

### imitation.scripts.ingredients.rl

This ingredient provides a reinforcement learning algorithm from stable-baselines3.

The algorithm instance is either freshly constructed or loaded from a file.

### Functions

| | |
|---|---|
| *config_hook*(config, command_name, logger) | Sets defaults equivalent to sb3.PPO default hyperparameters. |
| *load_rl_algo_from_path*(_seed, agent_path, ...) | **rtype**<br>BaseAlgorithm |
| *make_rl_algo*(venv, rl_cls, batch_size, ...) | Instantiates a Stable Baselines3 RL algorithm. |

imitation.scripts.ingredients.rl.**config_hook**(*config*, *command_name*, *logger*)

Sets defaults equivalent to sb3.PPO default hyperparameters.

This hook is a no-op if command_name is "sqil" (used only in train_imitation), which has its own config hook.

**Parameters**

- **config** – Sacred config dict.
- **command_name** – Sacred command name.
- **logger** – Sacred logger.

**Returns**

Updated Sacred config dict.

**Return type**

config

imitation.scripts.ingredients.rl.**load_rl_algo_from_path**(*_seed*, *agent_path*, *venv*, *rl_cls*, *rl_kwargs*, *relabel_reward_fn=None*)

**Return type**

BaseAlgorithm

imitation.scripts.ingredients.rl.**make_rl_algo**(*venv*, *rl_cls*, *batch_size*, *rl_kwargs*, *policy*, *_seed*, *relabel_reward_fn=None*)

Instantiates a Stable Baselines3 RL algorithm.

**Parameters**

- **venv** (VecEnv) – The vectorized environment to train on.
- **rl_cls** (Type[BaseAlgorithm]) – Type of a Stable Baselines3 RL algorithm.
- **batch_size** (int) – The batch size of the RL algorithm.
- **rl_kwargs** (Mapping[str, Any]) – Keyword arguments for RL algorithm constructor.
- **policy** (Mapping[str, Any]) – Configuration for the policy ingredient. We need the policy_cls and policy_kwargs component.
- **relabel_reward_fn** (Optional[*RewardFn*]) – Reward function used for reward relabeling in replay or rollout buffers of RL algorithms.

**Return type**

BaseAlgorithm

**Returns**

The RL algorithm.

**Raises**

- **ValueError** – *gen_batch_size* not divisible by *venv.num_envs*.
- **TypeError** – *rl_cls* is neither *OnPolicyAlgorithm* nor *OffPolicyAlgorithm*.

### imitation.scripts.ingredients.sqil

This ingredient provides a SQIL algorithm instance.

#### Functions

| | |
|---|---|
| *override_policy_cls*(config, command_name, logger) | |
| *override_rl_cls*(config, command_name, logger) | |

imitation.scripts.ingredients.sqil.**override_policy_cls**(*config*, *command_name*, *logger*)

imitation.scripts.ingredients.sqil.**override_rl_cls**(*config*, *command_name*, *logger*)

### imitation.scripts.ingredients.wb

This ingredient provides Weights & Biases logging.

#### Functions

| | |
|---|---|
| *wandb_init*(_run, wandb_name_prefix, ...) | Putting everything together to get the W&B kwargs for wandb.init(). |

imitation.scripts.ingredients.wb.**wandb_init**(*_run*, *wandb_name_prefix*, *wandb_tag*, *wandb_kwargs*, *wandb_additional_info*, *log_dir*)

Putting everything together to get the W&B kwargs for wandb.init().

**Parameters**

- **wandb_name_prefix** (str) – User-specified prefix for wandb run name.
- **wandb_tag** (Optional[str]) – User-specified tag for this run.
- **wandb_kwargs** (Mapping[str, Any]) – User-specified kwargs for wandb.init().
- **wandb_additional_info** (Mapping[str, Any]) – User-specific additional info to add to wandb experiment `config`.
- **log_dir** (str) – W&B logs will be stored in directory *{log_dir}/wandb/*.

**Raises**

**ModuleNotFoundError** – wandb is not installed.

**Return type**

None

### imitation.scripts.parallel

Runs a Sacred experiment in parallel.

#### Functions

| | |
|---|---|
| `main_console`() | |
| `parallel`(sacred_ex_name, run_name, ...) | Parallelize multiple runs of another Sacred Experiment using Ray Tune. |

imitation.scripts.parallel.**main_console**()

imitation.scripts.parallel.**parallel**(*sacred_ex_name*, *run_name*, *num_samples*, *search_space*, *base_named_configs*, *base_config_updates*, *resources_per_trial*, *init_kwargs*, *repeat*, *experiment_checkpoint_path*, *tune_run_kwargs*)

Parallelize multiple runs of another Sacred Experiment using Ray Tune.

A Sacred FileObserver is attached to the inner experiment and writes Sacred logs to "{RAY_LOCAL_DIR}/sacred/". These files are automatically copied over to *upload_dir* if that argument is provided in *tune_run_kwargs*.

**Parameters**

- **sacred_ex_name** (`str`) – The Sacred experiment to tune. Either "train_rl", "train_imitation", "train_adversarial" or "train_preference_comparisons".
- **run_name** (`str`) – A name describing this parallelizing experiment. This argument is also passed to *ray.tune.run* as the *name* argument. It is also saved in 'sacred/run.json' of each inner Sacred experiment under the 'experiment.name' key. This is equivalent to using the Sacred CLI '–name' option on the inner experiment. Offline analysis jobs can use this argument to group similar data.
- **num_samples** (`int`) – Number of times to sample from the hyperparameter space without considering repetition using *repeat*.
- **search_space** (`Mapping[str, Any]`) – A dictionary which can contain Ray Tune search objects like *ray.tune.grid_search* and *ray.tune.sample_from*, and is passed as the *config* argument to *ray.tune.run()*. After the *search_space* is transformed by Ray, it passed into *sacred_ex.run(**run_kwargs)* as *run_kwargs* (*sacred_ex* is the Sacred Experiment selected via *sacred_ex_name*). Usually *search_space* only has the keys "named_configs" and "config_updates", but any parameter names to *sacred.Experiment.run()* are okay.
- **base_named_configs** (`Sequence[str]`) – Default Sacred named configs. Any named configs taken from *search_space* are higher priority than the base_named_configs. Concretely, this priority is implemented by appending named configs taken from *search_space* to the run's named configs after *base_named_configs*. Named configs in *base_named_configs* don't appear in the automatically generated Ray directory name, unlike named configs from *search_space*.
- **base_config_updates** (`Mapping[str, Any]`) – Default Sacred config updates. Any config updates taken from *search_space* are higher priority than *base_config_updates*. Config updates in *base_config_updates* don't appear in the automatically generated Ray directory name, unlike config updates from *search_space*.
- **resources_per_trial** (`Mapping[str, Any]`) – Argument to *ray.tune.run()*.
- **init_kwargs** (`Mapping[str, Any]`) – Arguments to pass to *ray.init*.

- **repeat** (int) – Number of runs to repeat each trial for. If *repeat* > 1, then optuna is used as the default search algorithm unless specified otherwise in *tune_run_kwargs*.
- **experiment_checkpoint_path** (str) – Path containing the checkpoints of a previous experiment ran using this script. Useful for evaluating the best trial of the experiment.
- **tune_run_kwargs** (Dict[str, Any]) – Other arguments to pass to *ray.tune.run()*.

**Raises**

- **TypeError** – Named configs not string sequences or config updates not mappings.
- **ValueError** – *repeat* > 1 but *search_alg* is not an instance of *ray.tune.search.SearchAlgorithm*.

**Return type**

ExperimentAnalysis

**Returns**

The result of running the parallel experiment with *ray.tune.run()*. Useful for fetching the configs and results dataframe of all the trials.

### imitation.scripts.train_adversarial

Train GAIL or AIRL.

#### Functions

| | |
|---|---|
| *airl*() | |
| *gail*() | |
| *main_console*() | |
| *save*(trainer, save_path) | Save discriminator and generator. |
| *train_adversarial*(_run, show_config, ...) | Train an adversarial-network-based imitation learning algorithm. |

imitation.scripts.train_adversarial.**airl**()

imitation.scripts.train_adversarial.**gail**()

imitation.scripts.train_adversarial.**main_console**()

imitation.scripts.train_adversarial.**save**(*trainer*, *save_path*)

Save discriminator and generator.

imitation.scripts.train_adversarial.**train_adversarial**(*_run*, *show_config*, *algo_cls*, *algorithm_kwargs*, *total_timesteps*, *checkpoint_interval*, *agent_path*)

Train an adversarial-network-based imitation learning algorithm.

**Checkpoints:**

- **AdversarialTrainer train and test RewardNets are saved to**

  ***f"{log_dir}/checkpoints/{step}/reward_{train,test}.pt"***
  where step is either the training round or "final".

- Generator policies are saved to *f"{log_dir}/checkpoints/{step}/gen_policy/"*.

**Parameters**

- **`show_config`** (`bool`) – Print the merged config before starting training. This is analogous to the print_config command, but will show config after rather than before merging *algorithm_specific* arguments.
- **`algo_cls`** (`Type[AdversarialTrainer]`) – The adversarial imitation learning algorithm to use.
- **`algorithm_kwargs`** (`Mapping[str, Any]`) – Keyword arguments for the *GAIL* or *AIRL* constructor.
- **`total_timesteps`** (`int`) – The number of transitions to sample from the environment during training.
- **`checkpoint_interval`** (`int`) – Save the discriminator and generator models every *checkpoint_interval* rounds and after training is complete. If 0, then only save weights after training is complete. If <0, then don't save weights at all.
- **`agent_path`** (`Optional[str]`) – Path to a directory containing a pre-trained agent. If provided, then the agent will be initialized using this stored policy (warm start). If not provided, then the agent will be initialized using a random policy.

**Return type**

`Mapping[str, Mapping[str, float]]`

**Returns**

A dictionary with two keys. "imit_stats" gives the return value of *rollout_stats()* on rollouts test-reward-wrapped environment, using the final policy (remember that the ground-truth reward can be recovered from the "monitor_return" key). "expert_stats" gives the return value of *rollout_stats()* on the expert demonstrations.

## imitation.scripts.train_imitation

Trains DAgger on synthetic demonstrations generated from an expert policy.

### Functions

| | |
|---|---|
| *`bc`*(bc, _run, _rnd) | Runs BC training. |
| *`dagger`*(bc, dagger, _run, _rnd) | Runs DAgger training. |
| *`main_console`*() | |
| *`sqil`*(sqil, policy, rl, _run, _rnd) | **rtype**<br>`Mapping[str, Mapping[str, float]]` |

`imitation.scripts.train_imitation.`**`bc`**(*bc*, *_run*, *_rnd*)

Runs BC training.

**Parameters**

- **`bc`** (`Dict[str, Any]`) – Configuration for BC training.
- **`_run`** – Sacred run object.
- **`_rnd`** (`Generator`) – Random number generator provided by Sacred.

**Return type**
`Mapping[str, Mapping[str, float]]`

**Returns**
Statistics for rollouts from the trained policy and demonstration data.

`imitation.scripts.train_imitation.`**`dagger`**`(`*bc*, *dagger*, *_run*, *_rnd*`)`

Runs DAgger training.

**Parameters**

- **`bc`** (`Dict[str, Any]`) – Configuration for BC training.
- **`dagger`** (`Mapping[str, Any]`) – Arguments for DAgger training.
- **`_run`** – Sacred run object.
- **`_rnd`** (`Generator`) – Random number generator provided by Sacred.

**Return type**
`Mapping[str, Mapping[str, float]]`

**Returns**
Statistics for rollouts from the trained policy and demonstration data.

`imitation.scripts.train_imitation.`**`main_console`**`()`

`imitation.scripts.train_imitation.`**`sqil`**`(`*sqil*, *policy*, *rl*, *_run*, *_rnd*`)`

**Return type**
`Mapping[str, Mapping[str, float]]`

### imitation.scripts.train_preference_comparisons

Train a reward model using preference comparisons.

Can be used as a CLI script, or the *train_preference_comparisons* function can be called directly.

#### Functions

| | |
|---|---|
| *`main_console`*() | |
| *`save_checkpoint`*(trainer, save_path, ...) | Save reward model and optionally policy. |
| *`save_model`*(agent_trainer, save_path) | Save the model as *model.zip*. |
| *`train_preference_comparisons`*(...) | Train a reward model using preference comparisons. |

`imitation.scripts.train_preference_comparisons.`**`main_console`**`()`

`imitation.scripts.train_preference_comparisons.`**`save_checkpoint`**`(`*trainer*, *save_path*, *allow_save_policy*`)`

Save reward model and optionally policy.

`imitation.scripts.train_preference_comparisons.`**`save_model`**`(`*agent_trainer*, *save_path*`)`

Save the model as *model.zip*.

imitation.scripts.train_preference_comparisons.**train_preference_comparisons**(*total_timesteps*, *total_comparisons*, *num_iterations*, *comparison_queue_size*, *fragment_length*, *transition_oversampling*, *initial_comparison_frac*, *exploration_frac*, *trajectory_path*, *trajectory_generator_kwargs*, *save_preferences*, *agent_path*, *preference_model_kwargs*, *reward_trainer_kwargs*, *gatherer_cls*, *gatherer_kwargs*, *active_selection*, *active_selection_oversam-*

Train a reward model using preference comparisons.

**Parameters**

- **total_timesteps** (int) – number of environment interaction steps
- **total_comparisons** (int) – number of preferences to gather in total
- **num_iterations** (int) – number of times to train the agent against the reward model and then train the reward model against newly gathered preferences.
- **comparison_queue_size** (Optional[int]) – the maximum number of comparisons to keep in the queue for training the reward model. If None, the queue will grow without bound as new comparisons are added.
- **fragment_length** (int) – number of timesteps per fragment that is used to elicit preferences
- **transition_oversampling** (float) – factor by which to oversample transitions before creating fragments. Since fragments are sampled with replacement, this is usually chosen > 1 to avoid having the same transition in too many fragments.
- **initial_comparison_frac** (float) – fraction of total_comparisons that will be sampled before the rest of training begins (using the randomly initialized agent). This can be used to pretrain the reward model before the agent is trained on the learned reward.
- **exploration_frac** (float) – fraction of trajectory samples that will be created using partially random actions, rather than the current policy. Might be helpful if the learned policy explores too little and gets stuck with a wrong reward.
- **trajectory_path** (Optional[str]) – either None, in which case an agent will be trained and used to sample trajectories on the fly, or a path to a pickled sequence of TrajectoryWithRew to be trained on.
- **trajectory_generator_kwargs** (Mapping[str, Any]) – kwargs to pass to the trajectory generator.
- **save_preferences** (bool) – if True, store the final dataset of preferences to disk.
- **agent_path** (Optional[str]) – if given, initialize the agent using this stored policy rather than randomly.
- **preference_model_kwargs** (Mapping[str, Any]) – passed to PreferenceModel
- **reward_trainer_kwargs** (Mapping[str, Any]) – passed to BasicRewardTrainer or EnsembleRewardTrainer
- **gatherer_cls** (Type[*PreferenceGatherer*]) – type of PreferenceGatherer to use (defaults to SyntheticGatherer)
- **gatherer_kwargs** (Mapping[str, Any]) – passed to the PreferenceGatherer specified by gatherer_cls
- **active_selection** (bool) – use active selection fragmenter instead of random fragmenter
- **active_selection_oversampling** (int) – factor by which to oversample random fragments from the base fragmenter of active selection. this is usually chosen > 1 to allow the active selection algorithm to pick fragment pairs with highest uncertainty. = 1 implies no active selection.
- **uncertainty_on** (str) – passed to ActiveSelectionFragmenter

- **fragmenter_kwargs** (`Mapping[str, Any]`) – passed to RandomFragmenter
- **allow_variable_horizon** (`bool`) – If False (default), algorithm will raise an exception if it detects trajectories of different length during training. If True, overrides this safety check. WARNING: variable horizon episodes leak information about the reward via termination condition, and can seriously confound evaluation. Read https://imitation.readthedocs.io/en/latest/guide/variable_horizon.html before overriding this.
- **checkpoint_interval** (`int`) – Save the reward model and policy models (if trajectory_generator contains a policy) every *checkpoint_interval* iterations and after training is complete. If 0, then only save weights after training is complete. If <0, then don't save weights at all.
- **query_schedule** (`Union[str, Callable[[float], float]]`) – one of ("constant", "hyperbolic", "inverse_quadratic"). A function indicating how the total number of preference queries should be allocated to each iteration. "hyperbolic" and "inverse_quadratic" apportion fewer queries to later iterations when the policy is assumed to be better and more stable.
- **_rnd** (`Generator`) – Random number generator provided by Sacred.

**Return type**
`Mapping[str, Any]`

**Returns**
Rollout statistics from trained policy.

**Raises**
**ValueError** – Inconsistency between config and deserialized policy normalization.

### imitation.scripts.train_rl

Uses RL to train a policy from scratch, saving rollouts and policy.

**This can be used:**

1. To train a policy on a ground-truth reward function, as a source of synthetic "expert" demonstrations to train IRL or imitation learning algorithms.
2. To train a policy on a learned reward function, to solve a task or as a way of evaluating the quality of the learned reward function.

#### Functions

| | |
|---|---|
| *main_console*() | |
| *train_rl*(*, total_timesteps, ...) | Trains an expert policy from scratch and saves the rollouts and policy. |

imitation.scripts.train_rl.**main_console**()

imitation.scripts.train_rl.**train_rl**(*, *total_timesteps*, *normalize_reward*, *normalize_kwargs*, *reward_type*, *reward_path*, *load_reward_kwargs*, *rollout_save_final*, *rollout_save_n_timesteps*, *rollout_save_n_episodes*, *policy_save_interval*, *policy_save_final*, *agent_path*, *_rnd*)

Trains an expert policy from scratch and saves the rollouts and policy.

**Checkpoints:**

At applicable training steps *step* (where step is either an integer or "final"):

- Policies are saved to *{log_dir}/policies/{step}/*.
- Rollouts are saved to *{log_dir}/rollouts/{step}.npz*.

**Parameters**

- **total_timesteps** (int) – Number of training timesteps in *model.learn()*.
- **normalize_reward** (bool) – Applies normalization and clipping to the reward function by keeping a running average of training rewards. Note: this is may be redundant if using a learned reward that is already normalized.
- **normalize_kwargs** (dict) – kwargs for *VecNormalize*.
- **reward_type** (Optional[str]) – If provided, then load the serialized reward of this type, wrapping the environment in this reward. This is useful to test whether a reward model transfers. For more information, see *imitation.rewards.serialize.load_reward*.
- **reward_path** (Optional[str]) – A specifier, such as a path to a file on disk, used by reward_type to load the reward model. For more information, see *imitation.rewards.serialize.load_reward*.
- **load_reward_kwargs** (Optional[Mapping[str, Any]]) – Additional kwargs to pass to *predict_processed*. Examples are 'alpha' for :class: *AddSTDRewardWrapper* and 'update_stats' for :class: *NormalizedRewardNet*.
- **rollout_save_final** (bool) – If True, then save rollouts right after training is finished.
- **rollout_save_n_timesteps** (Optional[int]) – The minimum number of timesteps saved in every file. Could be more than *rollout_save_n_timesteps* because trajectories are saved by episode rather than by transition. Must set exactly one of *rollout_save_n_timesteps* and *rollout_save_n_episodes*.
- **rollout_save_n_episodes** (Optional[int]) – The number of episodes saved in every file. Must set exactly one of *rollout_save_n_timesteps* and *rollout_save_n_episodes*.
- **policy_save_interval** (int) – The number of training updates between in between intermediate rollout saves. If the argument is nonpositive, then don't save intermediate updates.
- **policy_save_final** (bool) – If True, then save the policy right after training is finished.
- **agent_path** (Optional[str]) – Path to load warm-started agent.
- **_rnd** (Generator) – Random number generator provided by Sacred.

**Return type**

Mapping[str, float]

**Returns**

The return value of *rollout_stats()* using the final policy.

### imitation.scripts.tuning

Tunes the hyperparameters of the algorithms.

#### Functions

| | |
|---|---|
| *evaluate_trial*(trial, num_eval_seeds, ...[, ...]) | Evaluate a given trial of a parallel run on a separate set of seeds. |
| *find_best_trial*(experiment_analysis, return_key) | Find the trial with the best mean return across all seeds. |
| *main_console*() | |
| *tune*(parallel_run_config[, ...]) | Tune hyperparameters of imitation algorithms using the parallel script. |

imitation.scripts.tuning.**evaluate_trial**(*trial*, *num_eval_seeds*, *run_name*, *parallel_run_config*, *resources_per_trial*, *return_key*, *print_return=False*)

Evaluate a given trial of a parallel run on a separate set of seeds.

**Parameters**

- **trial** (Trial) – The trial to evaluate.
- **num_eval_seeds** (int) – Number of distinct seeds to evaluate the best trial on.
- **run_name** (str) – The name of the evaluation run.
- **parallel_run_config** – Dictionary of arguments passed to the parallel script to get best_trial.
- **resources_per_trial** (Dict[str, int]) – Resources to be used for each evaluation trial.
- **return_key** (str) – The key of the return metric in the results dataframe.
- **print_return** (bool) – Whether to print the mean and std of the evaluation returns.

**Returns**

The result of the evaluation run.

**Return type**

eval_run

imitation.scripts.tuning.**find_best_trial**(*experiment_analysis*, *return_key*, *print_return=False*)

Find the trial with the best mean return across all seeds.

**Parameters**

- **experiment_analysis** (ExperimentAnalysis) – The result of a parallel/tuning experiment.
- **return_key** (str) – The key of the return metric in the results dataframe.
- **print_return** (bool) – Whether to print the mean and std of the returns of the best trial.

**Returns**

The trial with the best mean return across all seeds.

**Return type**

best_trial

imitation.scripts.tuning.**main_console**()

imitation.scripts.tuning.**tune**(*parallel_run_config*, *eval_best_trial_resource_multiplier=1*, *num_eval_seeds=5*)

Tune hyperparameters of imitation algorithms using the parallel script.

The parallel script is called twice in this function. The first call is to tune the hyperparameters. The second call is to evaluate the best trial on a separate set of seeds.

**Parameters**

- **parallel_run_config** – Dictionary of arguments to pass to the parallel script. This is used to define the search space for tuning the hyperparameters.
- **eval_best_trial_resource_multiplier** (int) – Factor by which to multiply the number of cpus per trial in *resources_per_trial*. This is useful for allocating more resources per trial to the evaluation trials than the resources for hyperparameter tuning since number of evaluation trials is usually much smaller than the number of tuning trials.
- **num_eval_seeds** (int) – Number of distinct seeds to evaluate the best trial on. Set to 0 to disable evaluation.

**Raises**

**ValueError** – If no trials are returned by the parallel run of tuning.

**Return type**

None

## 3.1.7 imitation.testing

Helper methods for unit tests.

May also be useful for users of imitation.

### Modules

| | |
|---|---|
| *imitation.testing.expert_trajectories* | Test utilities to conveniently generate expert trajectories. |
| *imitation.testing.hypothesis_strategies* | Hypothesis strategies for generating sequences of trajectories for testing. |
| *imitation.testing.reward_improvement* | Utility functions used to check if rewards improved wrt to previous rewards. |
| *imitation.testing.reward_nets* | Utility functions for testing reward nets. |

### imitation.testing.expert_trajectories

Test utilities to conveniently generate expert trajectories.

### Functions

| | |
|---|---|
| *generate_expert_trajectories*(env_id, ...) | Generate expert trajectories for the given environment. |
| *lazy_generate_expert_trajectories*(...) | Generate or load expert trajectories from cache. |
| *make_expert_transition_loader*(cache_dir, ...) | Creates different kinds of PyTorch data loaders for expert transitions. |

imitation.testing.expert_trajectories.**generate_expert_trajectories**(*env_id*, *num_trajectories*, *rng*)

Generate expert trajectories for the given environment.

Note: will just pull a pretrained policy from the Hugging Face model hub.

**Parameters**

- **env_id** (str) – The environment to generate trajectories for.
- **num_trajectories** (int) – The number of trajectories to generate.
- **rng** (Generator) – The random number generator to use.

**Return type**

Sequence[*TrajectoryWithRew*]

**Returns**

A list of trajectories with rewards.

imitation.testing.expert_trajectories.**lazy_generate_expert_trajectories**(*cache_path*, *env_id*, *num_trajectories*, *rng*)

Generate or load expert trajectories from cache.

**Parameters**

- **cache_path** (PathLike) – A path to the folder to be used as cache for the expert trajectories.
- **env_id** (str) – The environment to generate trajectories for.
- **num_trajectories** (int) – The number of trajectories to generate.
- **rng** (Generator) – The random number generator to use.

**Return type**

Sequence[*TrajectoryWithRew*]

**Returns**

A list of trajectories with rewards.

imitation.testing.expert_trajectories.**make_expert_transition_loader**(*cache_dir*, *batch_size*, *expert_data_type*, *env_name*, *rng*, *num_trajectories=1*, *shuffle=True*)

Creates different kinds of PyTorch data loaders for expert transitions.

**Parameters**

- **cache_dir** (Path) – The directory to use for caching the expert trajectories.
- **batch_size** (int) – The batch size to use for the data loader.
- **expert_data_type** (str) – The type of expert data to use. Can be one of "trajectories", "data_loader", "ducktyped_data_loader", "transitions".
- **env_name** (str) – The environment to generate trajectories for.
- **rng** (Generator) – The random number generator to use.
- **num_trajectories** (int) – The number of trajectories to generate.
- **shuffle** (bool) – Whether to shuffle the dataset when creating a data loader.

**Raises**

**ValueError** – If *expert_data_type* is not one of the supported types.

**Returns**

A pytorch data loader for expert transitions.

## imitation.testing.hypothesis_strategies

Hypothesis strategies for generating sequences of trajectories for testing.

### Module Attributes

| | |
|---|---|
| *gym_spaces* | A strategy to generate spaces supported by trajectory serialization. |
| *info_dict_contents* | A strategy to generate contents of the info dict for a trajectory. |
| *trajectory_length* | The length of a trajectories we want to test. |
| *trajectory* | A strategy to generate a single trajectory (with or without reward) for testing. |
| *trajectories_without_reward_list* | A strategy to generate lists of trajectories (without reward) for testing. |
| *trajectories_with_reward_list* | A strategy to generate lists of trajectories (with reward) for testing. |
| *trajectories_list* | A strategy to generate lists of trajectories (with or without reward) for testing. |

imitation.testing.hypothesis_strategies.**gym_spaces = sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)])**

A strategy to generate spaces supported by trajectory serialization.

imitation.testing.hypothesis_strategies.**info_dict_contents = dictionaries(keys=text(), values=one_of(integers(), floats(allow_nan=False), text(), lists(floats(allow_nan=False))))**

A strategy to generate contents of the info dict for a trajectory.

imitation.testing.hypothesis_strategies.**trajectories_list = one_of(lists(builds(_build_trajectory_without_reward, act_space=shared(sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), key='act_space'), info_dict_contents=dictionaries(keys=text(), values=one_of(integers(), floats(allow_nan=False), text(), lists(floats(allow_nan=False)))), length=integers(min_value=1, max_value=10), obs_space=shared(sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), key='obs_space'), terminal=booleans()), min_size=1, max_size=10), lists(builds(_build_trajectory_with_rew, act_space=shared(sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), key='act_space'), info_dict_contents=dictionaries(keys=text(), values=one_of(integers(), floats(allow_nan=False), text(), lists(floats(allow_nan=False)))), length=integers(min_value=1, max_value=10), max_rew=floats(min_value=-100, max_value=100), min_rew=floats(min_value=-100, max_value=100), obs_space=shared(sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), key='obs_space'), terminal=booleans()), min_size=1, max_size=10))**

A strategy to generate lists of trajectories (with or without reward) for testing.

All trajectories in the list are generated using the same spaces. They either all have reward or none of them have reward.

imitation.testing.hypothesis_strategies.**trajectories_with_reward_list = lists(builds(_build_trajectory_with_rew, act_space=shared(sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), key='act_space'), info_dict_contents=dictionaries(keys=text(), values=one_of(integers(), floats(allow_nan=False), text(), lists(floats(allow_nan=False)))), length=integers(min_value=1, max_value=10), max_rew=floats(min_value=-100, max_value=100), min_rew=floats(min_value=-100, max_value=100), obs_space=shared(sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), key='obs_space'), terminal=booleans()), min_size=1, max_size=10)**

A strategy to generate lists of trajectories (with reward) for testing.

All trajectories in the list are generated using the same spaces.

imitation.testing.hypothesis_strategies.**trajectories_without_reward_list = lists(builds(_build_trajectory_without_reward, act_space=shared(sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), key='act_space'), info_dict_contents=dictionaries(keys=text(), values=one_of(integers(), floats(allow_nan=False), text(), lists(floats(allow_nan=False)))), length=integers(min_value=1, max_value=10), obs_space=shared(sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), key='obs_space'), terminal=booleans()), min_size=1, max_size=10)**

A strategy to generate lists of trajectories (without reward) for testing.

All trajectories in the list are generated using the same spaces.

imitation.testing.hypothesis_strategies.**trajectory = one_of(builds(_build_trajectory_without_reward, act_space=sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), info_dict_contents=dictionaries(keys=text(), values=one_of(integers(), floats(allow_nan=False), text(), lists(floats(allow_nan=False)))), length=integers(min_value=1, max_value=10), obs_space=sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), terminal=booleans()), builds(_build_trajectory_with_rew, act_space=sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), info_dict_contents=dictionaries(keys=text(), values=one_of(integers(), floats(allow_nan=False), text(), lists(floats(allow_nan=False)))), length=integers(min_value=1, max_value=10), max_rew=floats(min_value=-100, max_value=100), min_rew=floats(min_value=-100, max_value=100), obs_space=sampled_from([Discrete(3), MultiDiscrete([3 4]), Box(-1.0, 1.0, (1,), float32), Box(-1.0, 1.0, (2,), float32), Box(-inf, inf, (2,), float32)]), terminal=booleans()))**

A strategy to generate a single trajectory (with or without reward) for testing.

imitation.testing.hypothesis_strategies.**trajectory_length = integers(min_value=1, max_value=10)**

The length of a trajectories we want to test.

### imitation.testing.reward_improvement

Utility functions used to check if rewards improved wrt to previous rewards.

#### Functions

| | |
|---|---|
| *is_significant_reward_improvement*(...[, p_value]) | Checks if the new rewards are really better than the old rewards. |
| *mean_reward_improved_by*(old_rews, new_rews, ...) | Checks if mean rewards improved wrt. |

imitation.testing.reward_improvement.**is_significant_reward_improvement**(*old_rewards*, *new_rewards*, *p_value=0.05*)

Checks if the new rewards are really better than the old rewards.

Ensures that this is not just due to lucky sampling by a permutation test.

**Parameters**

- **old_rewards** (Iterable[float]) – Iterable of "old" trajectory rewards (e.g. before training).
- **new_rewards** (Iterable[float]) – Iterable of "new" trajectory rewards (e.g. after training).

- **p_value** (float) – The maximum probability, that the old rewards are just as good as the new rewards, that we tolerate.

**Return type**
bool

**Returns**
True, if the new rewards are most probably better than the old rewards. For this, the probability, that the old rewards are just as good as the new rewards must be below *p_value*.

```
>>> is_significant_reward_improvement((5, 6, 7, 4, 4), (7, 5, 9, 9, 12))
True
```

```
>>> is_significant_reward_improvement((5, 6, 7, 4, 4), (7, 5, 9, 7, 4))
False
```

```
>>> is_significant_reward_improvement((5, 6, 7, 4, 4), (7, 5, 9, 7, 4), p_value=0.
↪3)
True
```

imitation.testing.reward_improvement.**mean_reward_improved_by**(*old_rews*, *new_rews*, *min_improvement*)

Checks if mean rewards improved wrt. to old rewards by a certain amount.

**Parameters**

- **old_rews** (Iterable[float]) – Iterable of "old" trajectory rewards (e.g. before training).
- **new_rews** (Iterable[float]) – Iterable of "new" trajectory rewards (e.g. after training).
- **min_improvement** (float) – The minimum amount of improvement that we expect.

**Returns**
True if the mean of the new rewards is larger than the mean of the old rewards by min_improvement.

```
>>> mean_reward_improved_by([5, 8, 7], [8, 9, 10], 2)
True
```

```
>>> mean_reward_improved_by([5, 8, 7], [8, 9, 10], 5)
False
```

### imitation.testing.reward_nets

Utility functions for testing reward nets.

### Functions

| | |
|---|---|
| *make_ensemble*(obs_space, action_space[, ...]) | Create a simple reward ensemble. |

### Classes

| | |
|---|---|
| *MockRewardNet*(observation_space, action_space) | A mock reward net for testing. |

**class** imitation.testing.reward_nets.**MockRewardNet**(*observation_space*, *action_space*, *value=0.0*)

Bases: *RewardNet*

A mock reward net for testing.

**__init__**(*observation_space*, *action_space*, *value=0.0*)

Create mock reward.

**Parameters**

- **observation_space** (Space) – observation space of the env
- **action_space** (Space) – action space of the env
- **value** (float) – The reward to always return. Defaults to 0.0.

**forward**(*state*, *action*, *next_state*, *done*)

Compute rewards for a batch of transitions and keep gradients.

**Return type**

Tensor

**training: bool**

imitation.testing.reward_nets.**make_ensemble**(*obs_space*, *action_space*, *num_members=2*, ***kwargs*)

Create a simple reward ensemble.

## 3.1.8 imitation.util

General utility functions: e.g. logging, configuration, etc.

### Modules

| | |
|---|---|
| *imitation.util.logger* | Logging for quantitative metrics and free-form text. |
| *imitation.util.networks* | Helper methods to build and run neural networks. |
| *imitation.util.registry* | Registry mapping IDs to objects, such as environments or policy loaders. |
| *imitation.util.sacred* | Helper methods for the *sacred* experimental configuration and logging framework. |
| *imitation.util.sacred_file_parsing* | Utilities to parse sacred run directories. |
| *imitation.util.util* | Miscellaneous utility methods. |
| *imitation.util.video_wrapper* | Wrapper to record rendered video frames from an environment. |

## imitation.util.logger

Logging for quantitative metrics and free-form text.

### Functions

| | |
|---|---|
| `configure`([folder, format_strs]) | Configure Stable Baselines logger to be *accumulate_means()*-compatible. |
| `make_output_format`(_format, log_dir[, ...]) | Returns a logger for the requested format. |

### Classes

| | |
|---|---|
| `HierarchicalLogger`(default_logger[, format_strs]) | A logger supporting contexts for accumulating mean values. |
| `WandbOutputFormat`() | A stable-baseline logger that writes to wandb. |

**class** imitation.util.logger.**HierarchicalLogger**(*default_logger*, *format_strs=('stdout', 'log', 'csv')*)

Bases: `Logger`

A logger supporting contexts for accumulating mean values.

*self.accumulate_means* creates a context manager. While in this context, values are loggged to a sub-logger, with only mean values recorded in the top-level (root) logger.

```
>>> import tempfile
>>> with tempfile.TemporaryDirectory() as dir:
...     logger: HierarchicalLogger = configure(dir, ('log',))
...     # record the key value pair (loss, 1.0) to path `dir`
...     # at step 1.
...     logger.record("loss", 1.0)
...     logger.dump(step=1)
...     with logger.accumulate_means("dataset"):
...         # record the key value pair `("raw/dataset/entropy", 5.0)` to path
...         # `dir/raw/dataset` at step 100
...         logger.record("entropy", 5.0)
...         logger.dump(step=100)
...         # record the key value pair `("raw/dataset/entropy", 6.0)` to path
...         # `dir/raw/dataset` at step 200
...         logger.record("entropy", 6.0)
...         logger.dump(step=200)
...     # record the key value pair `("mean/dataset/entropy", 5.5)` to path
...     # `dir` at step 1.
...     logger.dump(step=1)
...     with logger.add_accumulate_prefix("foo"), logger.accumulate_means("bar"):
...         # record the key value pair ("raw/foo/bar/biz", 42.0) to path
...         # `dir/raw/foo/bar` at step 2000
...         logger.record("biz", 42.0)
...         logger.dump(step=2000)
...     # record the key value pair `("mean/foo/bar/biz", 42.0)` to path
...     # `dir` at step 1.
...     logger.dump(step=1)
...     with open(os.path.join(dir, 'log.txt')) as f:
```

(continues on next page)

(continued from previous page)

```
...         print(f.read())
-------------------
| loss | 1        |
-------------------
---------------------------------
| mean/               |         |
|    dataset/entropy | 5.5      |
---------------------------------
-----------------------------
| mean/           |         |
|    foo/bar/biz | 42        |
-----------------------------
```

**__init__**(*default_logger*, *format_strs=('stdout', 'log', 'csv')*)

Builds HierarchicalLogger.

**Parameters**

- **default_logger** (`Logger`) – The default logger when not in an *accumulate_means* context. Also the logger to which mean values are written to after exiting from a context.
- **format_strs** (`Sequence[str]`) – A list of output format strings that should be used by every Logger initialized by this class during an *AccumulatingMeans* context. For details on available output formats see *stable_baselines3.logger.make_output_format*.

**accumulate_means**(*name*)

Temporarily modifies this HierarchicalLogger to accumulate means values.

Within this context manager, `self.record(key, value)` writes the "raw" values in `f"{self.default_logger.log_dir}/[{accumulate_prefix}/]{name}"` under the key `"raw/[{accumulate_prefix}/]{name}/[{key_prefix}/]{key}"`, where `accumulate_prefix` is the concatenation of all prefixes added by `add_accumulate_prefix` and `key_prefix` is the concatenation of all prefixes added by `add_key_prefix`, if any. At the same time, any call to `self.record` will also accumulate mean values on the default logger by calling:

```
self.default_logger.record_mean(
    f"mean/[{accumulate_prefix}/][{name}/][{key_prefix}/]{key}",
    value,
)
```

Multiple prefixes may be active at once. In this case the *prefix* is simply the concatenation of each of the active prefixes in the order they were created e.g. if the active prefixes are `['foo', 'bar']` then the prefix is `'foo/bar'`.

After the context exits, calling `self.dump()` will write the means of all the "raw" values accumulated during this context to `self.default_logger` under keys of the form `mean/{prefix}/{name}/{key}`

Note that the behavior of other logging methods, `log` and `record_mean` are unmodified and will go straight to the default logger.

**Parameters**

**name** (`str`) – A string key which determines the `folder` where raw data is written and temporary logging prefixes for raw and mean data. Entering an *accumulate_means* context in the future with the same *subdir* will safely append to logs written in this folder rather than overwrite.

**Yields**
None when the context is entered.

**Raises**
**`RuntimeError`** – If this context is entered into while already in an *accumulate_means* context.

**Return type**
`Generator`[`None`, `None`, `None`]

**`add_accumulate_prefix`**(*prefix*)

Add a prefix to the subdirectory used to accumulate means.

This prefix only applies when a *accumulate_means* context is active. If there are multiple active prefixes, then they are concatenated.

**Parameters**
**`prefix`** (`str`) – The prefix to add to the named sub.

**Yields**
None when the context manager is entered

**Raises**
**`RuntimeError`** – if accumulate means context is already active.

**Return type**
`Generator`[`None`, `None`, `None`]

**`add_key_prefix`**(*prefix*)

Add a prefix to the keys logged during an accumulate_means context.

This prefix only applies when a *accumulate_means* context is active. If there are multiple active prefixes, then they are concatenated.

**Parameters**
**`prefix`** (`str`) – The prefix to add to the keys.

**Yields**
None when the context manager is entered

**Raises**
**`RuntimeError`** – if accumulate means context is already active.

**Return type**
`Generator`[`None`, `None`, `None`]

**`close`**()

closes the file

**`current_logger: Optional[Logger]`**

**`default_logger: Logger`**

**`dump`**(*step=0*)

Write all of the diagnostics from the current iteration

**`format_strs: Sequence[str]`**

**`get_accumulate_prefixes`**()

**Return type**
`str`

**get_dir**()

Get directory that log files are being written to. will be None if there is no output directory (i.e., if you didn't call start)

**Return type**
`str`

**Returns**
the logging directory

**log**(**args*, ***kwargs*)

Write the sequence of args, with no separators, to the console and output files (if you've configured an output file).

**level: int. (see logger.py docs) If the global logger level is higher than**
the level argument here, don't print to stdout.

**Parameters**

- **args** – log the arguments
- **level** – the logging level (can be DEBUG=10, INFO=20, WARN=30, ERROR=40, DISABLED=50)

**record**(*key*, *val*, *exclude=None*)

Log a value of some diagnostic Call this once for each diagnostic quantity, each iteration If called many times, last value will be used.

**Parameters**

- **key** – save to log this key
- **value** – save to log this value
- **exclude** – outputs to be excluded

**record_mean**(*key*, *val*, *exclude=None*)

The same as record(), but if called many times, values averaged.

**Parameters**

- **key** – save to log this key
- **value** – save to log this value
- **exclude** – outputs to be excluded

**set_level**(*level*)

Set logging threshold on current logger.

**Parameters**
**level** (`int`) – the logging level (can be DEBUG=10, INFO=20, WARN=30, ERROR=40, DISABLED=50)

**Return type**
`None`

**class** imitation.util.logger.**WandbOutputFormat**

Bases: `KVWriter`

A stable-baseline logger that writes to wandb.

Users need to call *wandb.init()* before initializing *WandbOutputFormat*.

**__init__**()

Initializes an instance of WandbOutputFormat.

**Raises**

**ModuleNotFoundError** – wandb is not installed.

**close**()

Close owned resources

**Return type**

None

**write**(*key_values*, *key_excluded*, *step=0*)

Write a dictionary to file

**Parameters**

- **key_values** (Dict[str, Any]) –
- **key_excluded** (Dict[str, Tuple[str, ...]]) –
- **step** (int) –

**Return type**

None

imitation.util.logger.**configure**(*folder=None*, *format_strs=None*)

Configure Stable Baselines logger to be *accumulate_means()*-compatible.

After this function is called, *stable_baselines3.logger.{configure,reset}()* are replaced with stubs that raise RuntimeError.

**Parameters**

- **folder** (Union[str, bytes, PathLike, None]) – Argument from *stable_baselines3.logger.configure*.
- **format_strs** (Optional[Sequence[str]]) – An list of output format strings. For details on available output formats see *stable_baselines3.logger.make_output_format*.

**Return type**

*HierarchicalLogger*

**Returns**

The configured HierarchicalLogger instance.

imitation.util.logger.**make_output_format**(*_format*, *log_dir*, *log_suffix=''*, *max_length=50*)

Returns a logger for the requested format.

**Parameters**

- **_format** (str) – the requested format to log to ('stdout', 'log', 'json' or 'csv' or 'tensorboard').
- **log_dir** (str) – the logging directory.
- **log_suffix** (str) – the suffix for the log file.
- **max_length** (int) – the maximum length beyond which the keys get truncated.

**Return type**

KVWriter

**Returns**

the logger.

### imitation.util.networks

Helper methods to build and run neural networks.

#### Functions

| | |
|---|---|
| *build_cnn*(in_channels, hid_channels[, ...]) | Constructs a Torch CNN. |
| *build_mlp*(in_size, hid_sizes[, out_size, ...]) | Constructs a Torch MLP. |
| *training_mode*(m[, mode]) | Temporarily switch module `m` to specified training `mode`. |

#### Classes

| | |
|---|---|
| *BaseNorm*(num_features[, eps]) | Base class for layers that try to normalize the input to mean 0 and variance 1. |
| *EMANorm*(num_features[, decay, eps]) | Similar to RunningNorm but uses an exponential weighting. |
| *RunningNorm*(num_features[, eps]) | Normalizes input to mean 0 and standard deviation 1 using a running average. |
| *SqueezeLayer*(*args, **kwargs) | Torch module that squeezes a B*1 tensor down into a size-B vector. |

**class** imitation.util.networks.**BaseNorm**(*num_features*, *eps=1e-05*)

Bases: `Module`, `ABC`

Base class for layers that try to normalize the input to mean 0 and variance 1.

Similar to BatchNorm, LayerNorm, etc. but whereas they only use statistics from the current batch at train time, we use statistics from all batches.

**__init__**(*num_features*, *eps=1e-05*)

Builds RunningNorm.

**Parameters**

- **num_features** (`int`) – Number of features; the length of the non-batch dimension.
- **eps** (`float`) – Small constant for numerical stability. Inputs are rescaled by *1 / sqrt(estimated_variance + eps)*.

**count: Tensor**

**forward**(*x*)

Updates statistics if in training mode. Returns normalized *x*.

**Return type**

`Tensor`

**reset_running_stats**()

Resets running stats to defaults, yielding the identity transformation.

**Return type**

`None`

**running_mean: Tensor**

**running_var: Tensor**

**abstract update_stats**(*batch*)

Update *self.running_mean*, *self.running_var* and *self.count*.

**Return type**

None

**class** imitation.util.networks.**EMANorm**(*num_features*, *decay=0.99*, *eps=1e-05*)

Bases: *BaseNorm*

Similar to RunningNorm but uses an exponential weighting.

**__init__**(*num_features*, *decay=0.99*, *eps=1e-05*)

Builds EMARunningNorm.

**Parameters**

- **num_features** (int) – Number of features; the length of the non-batch dim.
- **decay** (float) – how quickly the weight on past samples decays over time.
- **eps** (float) – small constant for numerical stability.

**Raises**

**ValueError** – if decay is out of range.

**inv_learning_rate: Tensor**

**num_batches: IntTensor**

**reset_running_stats**()

Reset the running stats of the normalization layer.

**update_stats**(*batch*)

Update *self.running_mean* and *self.running_var* in batch mode.

Reference Algorithm 3 from: https://github.com/HumanCompatibleAI/imitation/files/9456540/Incremental_batch_EMA_and_EMV.pdf

**Parameters**

**batch** (Tensor) – A batch of data to use to update the running mean and variance.

**Return type**

None

**class** imitation.util.networks.**RunningNorm**(*num_features*, *eps=1e-05*)

Bases: *BaseNorm*

Normalizes input to mean 0 and standard deviation 1 using a running average.

Similar to BatchNorm, LayerNorm, etc. but whereas they only use statistics from the current batch at train time, we use statistics from all batches.

This should replicate the common practice in RL of normalizing environment observations, such as using VecNormalize in Stable Baselines. Note that the behavior of this class is slightly different from *VecNormalize*, e.g., it works with the current reward instead of return estimate, and subtracts the mean reward whereas VecNormalize only rescales it.

**count: Tensor**

**running_mean: Tensor**

**running_var: Tensor**

**training: bool**

**update_stats**(*batch*)

Update *self.running_mean*, *self.running_var* and *self.count*.

Uses Chan et al (1979), “Updating Formulae and a Pairwise Algorithm for Computing Sample Variances.” to update the running moments in a numerically stable fashion.

**Parameters**

**batch** (`Tensor`) – A batch of data to use to update the running mean and variance.

**Return type**

`None`

**class** imitation.util.networks.**SqueezeLayer**(**args*, ***kwargs*)

Bases: `Module`

Torch module that squeezes a B*1 tensor down into a size-B vector.

**forward**(*x*)

Define 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**

imitation.util.networks.**build_cnn**(*in_channels*, *hid_channels*, *out_size=1*, *name=None*, *activation=<class 'torch.nn.modules.activation.ReLU'>*, *kernel_size=3*, *stride=1*, *padding='same'*, *dropout_prob=0.0*, *squeeze_output=False*)

Constructs a Torch CNN.

**Parameters**

- **in_channels** (`int`) – number of channels of individual inputs; input to the CNN will have shape (batch_size, in_size, in_height, in_width).
- **hid_channels** (`Iterable[int]`) – number of channels of hidden layers. If this is an empty iterable, then we build a linear function approximator.
- **out_size** (`int`) – size of output vector.
- **name** (`Optional[str]`) – Name to use as a prefix for the layers ID.
- **activation** (`Type[Module]`) – activation to apply after hidden layers.
- **kernel_size** (`int`) – size of convolutional kernels.
- **stride** (`int`) – stride of convolutional kernels.
- **padding** (`Union[int, str]`) – padding of convolutional kernels.
- **dropout_prob** (`float`) – Dropout probability to use after each hidden layer. If 0, no dropout layers are added to the network.

- **squeeze_output** (bool) – if out_size=1, then squeeze_input=True ensures that CNN output is of size (B,) instead of (B,1).

**Returns**

**a CNN mapping from inputs of size (batch_size, in_size, in_height,**
in_width) to (batch_size, out_size), unless out_size=1 and squeeze_output=True, in which case the output is of size (batch_size, ).

**Return type**
nn.Module

**Raises**
**ValueError** – if squeeze_output was supplied with out_size!=1.

imitation.util.networks.**build_mlp**(*in_size*, *hid_sizes*, *out_size=1*, *name=None*, *activation=<class 'torch.nn.modules.activation.ReLU'>*, *dropout_prob=0.0*, *squeeze_output=False*, *flatten_input=False*, *normalize_input_layer=None*)

Constructs a Torch MLP.

**Parameters**

- **in_size** (int) – size of individual input vectors; input to the MLP will be of shape (batch_size, in_size).
- **hid_sizes** (Iterable[int]) – sizes of hidden layers. If this is an empty iterable, then we build a linear function approximator.
- **out_size** (int) – size of output vector.
- **name** (Optional[str]) – Name to use as a prefix for the layers ID.
- **activation** (Type[Module]) – activation to apply after hidden layers.
- **dropout_prob** (float) – Dropout probability to use after each hidden layer. If 0, no dropout layers are added to the network.
- **squeeze_output** (bool) – if out_size=1, then squeeze_input=True ensures that MLP output is of size (B,) instead of (B,1).
- **flatten_input** (bool) – should input be flattened along axes 1, 2, 3, …? Useful if you want to, e.g., process small images inputs with an MLP.
- **normalize_input_layer** (Optional[Type[Module]]) – if specified, module to use to normalize inputs; e.g. *nn.BatchNorm* or *RunningNorm*.

**Returns**

**an MLP mapping from inputs of size (batch_size, in_size) to**
(batch_size, out_size), unless out_size=1 and squeeze_output=True, in which case the output is of size (batch_size, ).

**Return type**
nn.Module

**Raises**
**ValueError** – if squeeze_output was supplied with out_size!=1.

imitation.util.networks.**evaluating**(*m: Module*, ***, *mode: bool = False*)

Temporarily switch module `m` to specified training `mode`.

**Parameters**

- **m** – The module to switch the mode of.
- **mode** – whether to set training mode (`True`) or evaluation (`False`).

**Yields**

The module *m*.

`imitation.util.networks.`**`training`**(*m: Module*, *, *mode: bool = True*)

Temporarily switch module `m` to specified training `mode`.

**Parameters**

- **m** – The module to switch the mode of.
- **mode** – whether to set training mode (`True`) or evaluation (`False`).

**Yields**

The module *m*.

`imitation.util.networks.`**`training_mode`**(*m*, *mode=False*)

Temporarily switch module `m` to specified training `mode`.

**Parameters**

- **m** (`Module`) – The module to switch the mode of.
- **mode** (`bool`) – whether to set training mode (`True`) or evaluation (`False`).

**Yields**

The module *m*.

### imitation.util.registry

Registry mapping IDs to objects, such as environments or policy loaders.

#### Module Attributes

| | |
|---|---|
| *LoaderFn* | The type stored in Registry is commonly an instance of LoaderFn. |

#### Functions

| | |
|---|---|
| *build_loader_fn_require_env*(fn, **kwargs) | Converts a factory taking an environment into a LoaderFn. |
| *build_loader_fn_require_space*(fn, **kwargs) | Converts a factory taking observation and action space into a LoaderFn. |
| *load_attr*(name) | Load an attribute in format path.to.module:attribute. |

### Classes

| | |
|---|---|
| *Registry*() | A registry mapping IDs to type T objects, with support for lazy loading. |

imitation.util.registry.**LoaderFn**

The type stored in Registry is commonly an instance of LoaderFn.

alias of `Callable`[[...], `T`]

**class** imitation.util.registry.**Registry**

Bases: `Generic`[`T`]

A registry mapping IDs to type T objects, with support for lazy loading.

The registry allows for insertion and retrieval. Modification of existing elements is not allowed.

If the registered item is a string, it is assumed to be a path to an attribute in the form path.to.module:attribute. In this case, the module is loaded only if and when the registered item is retrieved.

This is helpful both to reduce overhead from importing unused modules, and when some modules may have additional dependencies that are not installed in all deployments.

Note: This is a similar idea to gym.EnvRegistry.

**__init__**()

Builds empty Registry.

**get**(*key*)

**Return type**

`TypeVar`(`T`)

**keys**()

**Return type**

`Iterable`[`str`]

**register**(*key*, ***, *value=None*, *indirect=None*)

imitation.util.registry.**build_loader_fn_require_env**(*fn*, ***kwargs*)

Converts a factory taking an environment into a LoaderFn.

**Return type**

`Callable`[`...`, `TypeVar`(`T`)]

imitation.util.registry.**build_loader_fn_require_space**(*fn*, ***kwargs*)

Converts a factory taking observation and action space into a LoaderFn.

**Return type**

`Callable`[`...`, `TypeVar`(`T`)]

imitation.util.registry.**load_attr**(*name*)

Load an attribute in format path.to.module:attribute.

### imitation.util.sacred

Helper methods for the *sacred* experimental configuration and logging framework.

#### Functions

| | |
|---|---|
| `build_sacred_symlink`(log_dir, run) | Constructs a symlink "{log_dir}/sacred" => "${SACRED_PATH}". |
| `dict_get_nested`(d, nested_key, *[, sep, default]) | **rtype**<br>`Any` |
| `dir_contains_sacred_jsons`(dir_path) | **rtype**<br>`bool` |
| `filter_subdirs`(root_dir[, filter_fn, nested_ok]) | Walks through a directory tree, returning paths to filtered subdirectories. |
| `get_sacred_dir_from_run`(run) | Returns path to the sacred directory, or None if not found. |

#### Classes

| | |
|---|---|
| `SacredDicts`(sacred_dir, config, run) | Each dict *foo* is loaded from *f"{sacred_dir}/foo.json"*. |

**class** `imitation.util.sacred.`**`SacredDicts`**(*sacred_dir: Path*, *config: dict*, *run: dict*)

Bases: `tuple`

Each dict *foo* is loaded from *f"{sacred_dir}/foo.json"*.

**config: dict**

**classmethod load_from_dir**(*sacred_dir*)

**run: dict**

**sacred_dir: Path**

`imitation.util.sacred.`**`build_sacred_symlink`**(*log_dir*, *run*)

Constructs a symlink "{log_dir}/sacred" => "${SACRED_PATH}".

**Return type**
`None`

`imitation.util.sacred.`**`dict_get_nested`**(*d*, *nested_key*, ***, *sep='.'*, *default=None*)

**Return type**
`Any`

`imitation.util.sacred.`**`dir_contains_sacred_jsons`**(*dir_path*)

**Return type**
`bool`

imitation.util.sacred.**filter_subdirs**(*root_dir*, *filter_fn=<function dir_contains_sacred_jsons>*, *, *nested_ok=False*)

Walks through a directory tree, returning paths to filtered subdirectories.

Does not follow symlinks.

**Parameters**

- **root_dir** (Path) – The start of the directory tree walk.
- **filter_fn** (Callable[[Path], bool]) – A function with takes a directory path and returns True if we should include the directory path in this function's return value.
- **nested_ok** (bool) – Allow returning "nested" directories, i.e. a return value where some elements are subdirectories of other elements.

**Return type**

Sequence[Path]

**Returns**

A list of all subdirectory paths where *filter_fn(path) == True*.

**Raises**

**ValueError** – If *nested_ok* is False and one of the filtered directory paths is a subdirecotry of another.

imitation.util.sacred.**get_sacred_dir_from_run**(*run*)

Returns path to the sacred directory, or None if not found.

**Return type**

Optional[Path]

### imitation.util.sacred_file_parsing

Utilities to parse sacred run directories.

### Functions

| | |
|---|---|
| *find_sacred_runs*(run_path[, only_completed_runs]) | Recursively iterates the sacred runs found below the given path. |
| *group_runs_by_algo_and_env*(path[, ...]) | Groups the runs found below the given path by algorithm and environment. |

imitation.util.sacred_file_parsing.**find_sacred_runs**(*run_path*, *only_completed_runs=False*)

Recursively iterates the sacred runs found below the given path.

Assumes runs in the format of the sacred FileStorageObserver: each run consists of a folder that contains a config.json and a run.json file.

Note: will work with nested directories and can therefore be applied to the *output/sacred* folder of the command line interface which creates sub-folders for each script.

**Parameters**

- **run_path** (Path) – The path to search for sacred run directories.
- **only_completed_runs** (bool) – If True, only yields runs that have a run.json file with status "COMPLETED".

**Yields**
Tuples of (config, run) dicts.

**Return type**
`Generator[Tuple[Dict[str, Any], Dict[str, Any]], None, None]`

imitation.util.sacred_file_parsing.**group_runs_by_algo_and_env**(*path*, *only_completed_runs=False*)

Groups the runs found below the given path by algorithm and environment.

Access all the runs of algorithm *algo* and environment *env* via *runs_by_algo_and_env[algo][env]*.

**Parameters**

- **path** (`Path`) – The path to search for sacred run directories.
- **only_completed_runs** (`bool`) – If True, only yields runs that have a run.json file with status "COMPLETED".

**Return type**
`Dict[str, Dict[str, List[Dict[str, Any]]]]`

**Returns**
A dictionary mapping algorithms to environments to lists of runs.

### imitation.util.util

Miscellaneous utility methods.

#### Functions

| | |
|---|---|
| *clear_screen*() | Clears the console screen. |
| *docstring_parameter*(*args, **kwargs) | Treats the docstring as a format string, substituting in the arguments. |
| *endless_iter*(iterable) | Generator that endlessly yields elements from *iterable*. |
| *get_first_iter_element*(iterable) | Get first element of an iterable and a new fresh iterable. |
| *make_seeds*() | Generate n random seeds from a random state. |
| *make_unique_timestamp*() | Timestamp, with random uuid added to avoid collisions. |
| *make_vec_env*(env_name, *, rng[, n_envs, ...]) | Makes a vectorized environment. |
| *oric*(x) | Optimal rounding under integer constraints. |
| *parse_optional_path*(path[, allow_relative, ...]) | Parse an optional path to a *pathlib.Path* object. |
| *parse_path*(path[, allow_relative, ...]) | Parse a path to a *pathlib.Path* object. |
| *safe_to_numpy*() | Convert torch tensor to numpy. |
| *safe_to_tensor*(array, **kwargs) | Converts a NumPy array to a PyTorch tensor. |
| *save_policy*(policy, policy_path) | Save policy to a path. |
| *split_in_half*(x) | Split an integer in half, rounding up. |
| *tensor_iter_norm*(tensor_iter[, ord]) | Compute the norm of a big vector that is produced one tensor chunk at a time. |

imitation.util.util.**clear_screen**()

Clears the console screen.

**Return type**
`None`

imitation.util.util.**docstring_parameter**(**args*, ***kwargs*)

Treats the docstring as a format string, substituting in the arguments.

imitation.util.util.**endless_iter**(*iterable*)

Generator that endlessly yields elements from *iterable*.

```
>>> x = range(2)
>>> it = endless_iter(x)
>>> next(it)
0
>>> next(it)
1
>>> next(it)
0
```

**Parameters**

**iterable** (Iterable[TypeVar(T)]) – The non-iterator iterable object to endlessly iterate over.

**Return type**

Iterator[TypeVar(T)]

**Returns**

An iterator that repeats the elements in *iterable* forever.

**Raises**

**ValueError** – if iterable is an iterator – that will be exhausted, so cannot be iterated over endlessly.

imitation.util.util.**get_first_iter_element**(*iterable*)

Get first element of an iterable and a new fresh iterable.

The fresh iterable has the first element added back using `itertools.chain`. If the iterable is not an iterator, this is equivalent to `(next(iter(iterable)), iterable)`.

**Parameters**

**iterable** (Iterable[TypeVar(T)]) – The iterable to get the first element of.

**Return type**

Tuple[TypeVar(T), Iterable[TypeVar(T)]]

**Returns**

A tuple containing the first element of the iterable, and a fresh iterable with all the elements.

**Raises**

**ValueError** – *iterable* is empty – the first call to it returns no elements.

imitation.util.util.**make_seeds**(*rng: Generator*) → int

imitation.util.util.**make_seeds**(*rng: Generator*, *n: int*) → List[int]

Generate n random seeds from a random state.

**Parameters**

- **rng** (Generator) – The random state to use to generate seeds.
- **n** (Optional[int]) – The number of seeds to generate.

**Return type**

Union[Sequence[int], int]

**Returns**
A list of n random seeds.

`imitation.util.util.`**`make_unique_timestamp`**`()`

Timestamp, with random uuid added to avoid collisions.

**Return type**
`str`

`imitation.util.util.`**`make_vec_env`**`(`*env_name*, ***, *rng*, *n_envs=8*, *parallel=False*, *log_dir=None*, *max_episode_steps=None*, *post_wrappers=None*, *env_make_kwargs=None*`)`

Makes a vectorized environment.

**Parameters**

- **`env_name`** (`str`) – The Env's string id in Gym.
- **`rng`** (`Generator`) – The random state to use to seed the environment.
- **`n_envs`** (`int`) – The number of duplicate environments.
- **`parallel`** (`bool`) – If True, uses SubprocVecEnv; otherwise, DummyVecEnv.
- **`log_dir`** (`Optional[str]`) – If specified, saves Monitor output to this directory.
- **`max_episode_steps`** (`Optional[int]`) – If specified, wraps each env in a TimeLimit wrapper with this episode length. If not specified and *max_episode_steps* exists for this *env_name* in the Gym registry, uses the registry *max_episode_steps* for every TimeLimit wrapper (this automatic wrapper is the default behavior when calling *gym.make*). Otherwise the environments are passed into the VecEnv unwrapped.
- **`post_wrappers`** (`Optional[Sequence[Callable[[Env, int], Env]]]`) – If specified, iteratively wraps each environment with each of the wrappers specified in the sequence. The argument should be a Callable accepting two arguments, the Env to be wrapped and the environment index, and returning the wrapped Env.
- **`env_make_kwargs`** (`Optional[Mapping[str, Any]]`) – The kwargs passed to *spec.make*.

**Return type**
`VecEnv`

**Returns**
A VecEnv initialized with *n_envs* environments.

`imitation.util.util.`**`oric`**`(`*x*`)`

Optimal rounding under integer constraints.

Given a vector of real numbers such that the sum is an integer, returns a vector of rounded integers that preserves the sum and which minimizes the Lp-norm of the difference between the rounded and original vectors for all p >= 1. Algorithm from https://arxiv.org/abs/1501.00014. Runs in O(n log n) time.

**Parameters**
**`x`** (`ndarray`) – A 1D vector of real numbers that sum to an integer.

**Return type**
`ndarray`

**Returns**
A 1D vector of rounded integers, preserving the sum.

imitation.util.util.**parse_optional_path**(*path*, *allow_relative=True*, *base_directory=None*)

Parse an optional path to a *pathlib.Path* object.

All resulting paths are resolved, absolute paths. If *allow_relative* is True, then relative paths are allowed as input, and are resolved relative to the current working directory, or relative to *base_directory* if it is specified.

**Parameters**

- **path** (Union[str, bytes, PathLike, None]) – The path to parse. Can be a string, bytes, or *os.PathLike*.
- **allow_relative** (bool) – If True, then relative paths are allowed as input, and are resolved relative to the current working directory. If False, an error is raised if the path is not absolute.
- **base_directory** (Optional[Path]) – If specified, then relative paths are resolved relative to this directory, instead of the current working directory.

**Return type**

Optional[Path]

**Returns**

A *pathlib.Path* object, or None if *path* is None.

imitation.util.util.**parse_path**(*path*, *allow_relative=True*, *base_directory=None*)

Parse a path to a *pathlib.Path* object.

All resulting paths are resolved, absolute paths. If *allow_relative* is True, then relative paths are allowed as input, and are resolved relative to the current working directory, or relative to *base_directory* if it is specified.

**Parameters**

- **path** (Union[str, bytes, PathLike]) – The path to parse. Can be a string, bytes, or *os.PathLike*.
- **allow_relative** (bool) – If True, then relative paths are allowed as input, and are resolved relative to the current working directory. If False, an error is raised if the path is not absolute.
- **base_directory** (Optional[Path]) – If specified, then relative paths are resolved relative to this directory, instead of the current working directory.

**Return type**

Path

**Returns**

A *pathlib.Path* object.

**Raises**

- **ValueError** – If *allow_relative* is False and the path is not absolute.
- **ValueError** – If *base_directory* is specified and *allow_relative* is False.

imitation.util.util.**safe_to_numpy**(*obj: Union[ndarray, Tensor]*, *warn: bool = False*) → ndarray

imitation.util.util.**safe_to_numpy**(*obj: None*, *warn: bool = False*) → None

Convert torch tensor to numpy.

If the object is already a numpy array, return it as is. If the object is none, returns none.

**Parameters**

- **obj** (Union[ndarray, Tensor, None]) – torch tensor object to convert to numpy array

- **warn** (bool) – if True, warn if the object is not already a numpy array. Useful for warning the user of a potential performance hit if a torch tensor is not the expected input type.

**Return type**
Optional[ndarray]

**Returns**
Object converted to numpy array

imitation.util.util.**safe_to_tensor**(*array*, ***kwargs*)

Converts a NumPy array to a PyTorch tensor.

The data is copied in the case where the array is non-writable. Unfortunately if you just use *th.as_tensor* for this, an ugly warning is logged and there's undefined behavior if you try to write to the tensor.

**Parameters**

- **array** (Union[ndarray, Tensor]) – The array to convert to a PyTorch tensor.
- **kwargs** – Additional keyword arguments to pass to *th.as_tensor*.

**Return type**
Tensor

**Returns**
A PyTorch tensor with the same content as *array*.

imitation.util.util.**save_policy**(*policy*, *policy_path*)

Save policy to a path.

**Parameters**

- **policy** (BasePolicy) – policy to save.
- **policy_path** (Union[str, bytes, PathLike]) – path to save policy to.

**Return type**
None

imitation.util.util.**split_in_half**(*x*)

Split an integer in half, rounding up.

This is to ensure that the two halves sum to the original integer.

**Parameters**
**x** (int) – The integer to split.

**Return type**
Tuple[int, int]

**Returns**
A tuple containing the two halves of *x*.

imitation.util.util.**tensor_iter_norm**(*tensor_iter*, *ord=2*)

Compute the norm of a big vector that is produced one tensor chunk at a time.

**Parameters**

- **tensor_iter** (Iterable[Tensor]) – an iterable that yields tensors.
- **ord** (Union[int, float]) – order of the p-norm (can be any int or float except 0 and NaN).

**Return type**
Tensor

**Returns**
Norm of the concatenated tensors.

**Raises**
**ValueError** – ord is 0 (unsupported).

## imitation.util.video_wrapper

Wrapper to record rendered video frames from an environment.

### Classes

| | |
|---|---|
| *VideoWrapper*(env, directory[, single_video]) | Creates videos from wrapped environment by calling render after each timestep. |

**class** imitation.util.video_wrapper.**VideoWrapper**(*env*, *directory*, *single_video=True*)

Bases: Wrapper

Creates videos from wrapped environment by calling render after each timestep.

**__init__**(*env*, *directory*, *single_video=True*)

Builds a VideoWrapper.

**Parameters**

- **env** (Env) – the wrapped environment.
- **directory** (Path) – the output directory.
- **single_video** (bool) – if True, generates a single video file, with episodes concatenated. If False, a new video file is created for each episode. Usually a single video file is what is desired. However, if one is searching for an interesting episode (perhaps by looking at the metadata), then saving to different files can be useful.

**close**()

Closes the wrapper and env.

**Return type**
None

**directory: Path**

**episode_id: int**

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

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

**Return type**
Tuple[TypeVar(WrapperObsType), Dict[str, Any]]

**single_video: bool**

**step**(*action*)

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

**Return type**
Tuple[TypeVar(WrapperObsType), SupportsFloat, bool, bool, Dict[str, Any]]

**video_recorder: Optional[VideoRecorder]**

## 3.2 Developer Guide

This guide explains the library structure of imitation. The code is organized such that logically similar files are grouped into a subpackage. We maintain the following subpackages in `src/imitation`:

- `algorithms`: the core implementation of imitation and reward learning algorithms.
- `data`: modules to collect, store and manipulate transitions and trajectories from RL environments.
- `envs`: provides test environments.
- `policies`: provides modules that define policies and methods to manipulate them (e.g., serialization).
- `regularization`: implements a variety of regularization techniques for NN weights.
- `rewards`: modules to build, serialize and preprocess neural network based reward functions.
- `scripts`: command-line scripts for running experiments through Sacred.
- `util`: provides utility functions like logging, configurations, etc.

### 3.2.1 Algorithms

The `imitation.algorithms.base` module defines the following two classes:

- `BaseImitationAlgorithm`: Base class for all imitation algorithms.
- `DemonstrationAlgorithm`: Base class for all demonstration-based algorithms like BC, IRL, etc. This class subclasses `BaseImitationAlgorithm`.

  Demonstration algorithms offer the following methods and properties:

  - `policy` property that returns a policy imitating the demonstration data.
  - `set_demonstrations` method that sets the demonstrations data for learning.

All of the algorithms provide the `train` method for training an agent and/or a reward network.

All the available algorithms are present in `algorithms/` with each algorithm in a distinct file. Adversarial algorithms like AIRL and GAIL are present in `algorithms/adversarial`.

### 3.2.2 Data

Modules handling environment data.

For example: types for transitions/trajectories; methods to compute rollouts; buffers to store transitions; helpers for these modules.

`data.wrapper.BufferingWrapper`: Wraps a vectorized environment `VecEnv` to save the trajectories from all the environments in a buffer.

`data.wrapper.RolloutInfoWrapper`: Wraps a `gym.Env` environment to log the original observations and rewards received from the environment. The observations and rewards of the entire episode are logged in the `info` dictionary with the key `"rollout"`, in the final time step of the episode. This wrapper is useful for saving rollout trajectories, especially in cases where you want to bypass the reward and/or observation overrides from other wrappers. See `data.rollout.unwrap_traj` for details and `scripts/train_rl.py` for an example use case.

`data.rollout.rollout`: Generates rollout by taking in any policy as input along with the environment.

### 3.2.3 Policies

The `imitation.policies` subpackage contains the following modules:

- `policies.base`: defines commonly used policies across the library like `FeedForward32Policy`, `SAC1024Policy`, `NormalizeFeaturesExtractor`, etc.
- `policies.exploration_wrapper`: defines the `ExplorationWrapper` class that wraps a policy to create a partially randomized policy useful for exploration.
- `policies.replay_buffer_wrapper`: defines the `ReplayBufferRewardWrapper` to wrap a replay buffer that returns transitions with rewards specified by a reward function.
- `policies.serialize`: defines various functions to save and load serialized policies from the disk or the Hugging Face hub.

### 3.2.4 Regularization

The `imitation.regularization` subpackage provides an API for creating neural network regularizers. It provides classes such as `regularizers.LpRegularizer` and `regularizers.WeightDecayRegularizer` to regularize the loss function and the weights of a network, respectively. The `updaters.IntervalParamScaler` class also provides support to scale the lambda hyperparameter of a regularizer up when the ratio of validation to training loss is above an upper bound, and scales it down when the ratio drops below a lower bound.

### 3.2.5 Rewards

The `imitation.rewards` subpackage contains code related to building, serializing, and loading reward networks. Some of the classes include:

- `rewards.reward_nets.RewardNet`: is the base reward network class. Reward networks can take state, action, and the next state as input to predict the reward. The `forward` method is used while training the network, whereas the `predict` method is used during evaluation.
- `rewards.reward_nets.BasicRewardNet`: builds a MLP reward network.
- `rewards.reward_nets.CnnRewardNet`: builds a CNN based reward network.
- `rewards.reward_nets.RewardEnsemble`: builds an ensemble of reward networks.
- `rewards.reward_wrapper.RewardVecEnvWrapper`: This class wraps a `VecEnv` with a custom `RewardFn`. The default reward function of the environment is overridden with the passed reward function, and the original rewards are stored in the `info_dict` with the `original_env_rew` key. This class is used to override the original reward function of an environment with a learned reward function from the reward learning algorithms like preference comparisons.

The `imitation.rewards.serialize` module contains functions to load serialized reward functions.

For more see the *Reward Networks Tutorial*.

### 3.2.6 Scripts

We use Sacred to provide a command-line interface to run the experiments. The scripts to run the end-to-end experiments are available in `scripts/`. You can take a look at the following doc links to understand how to use Sacred:

- Experiment Overview: Explains how to create and run experiments. Each script, defined in `scripts/`, has a corresponding experiment object, defined in `scripts/config`, with the experiment object and Python source files named after the algorithm(s) supported. For example, the `train_rl_ex` object is defined in `scripts.config.train_rl` and its main function is in `scripts.train_rl`.
- Ingredients: Explains how to use ingredients to avoid code duplication across experiments. The ingredients used in our experiments are defined in `scripts/ingredients/`:

| | |
|---|---|
| *imitation.scripts.ingredients.logging* | This ingredient provides a number of logging utilities. |
| *imitation.scripts.ingredients.demonstrations* | This ingredient provides (expert) demonstrations to learn from. |
| *imitation.scripts.ingredients.environment* | This ingredient provides a vectorized gym environment. |
| *imitation.scripts.ingredients.expert* | This ingredient provides an expert policy. |
| *imitation.scripts.ingredients.reward* | This ingredient provides a reward network. |
| *imitation.scripts.ingredients.rl* | This ingredient provides a reinforcement learning algorithm from stable-baselines3. |
| *imitation.scripts.ingredients.policy* | This ingredient provides a newly constructed stable-baselines3 policy. |
| *imitation.scripts.ingredients.wb* | This ingredient provides Weights & Biases logging. |

- Configurations: Explains how to use configurations to parametrize runs. The configurations for different algorithms are defined in their file in `scripts/`. Some of the commonly used configs and ingredients used across algorithms are defined in `scripts/ingredients/`.
- Command-Line Interface: Explains how to run the experiments through the command-line interface. Also, note the section on how to print configs to verify the configurations used for the run.
- Controlling Randomness: Explains how to control randomness by seeding experiments through Sacred.

### 3.2.7 Util

`imitation.util.logger.HierarchicalLogger`: A logger that supports contexts for accumulating the mean of values of all the logged keys. The logger internally maintains one separate `stable_baselines3.common.logger.Logger` object for logging the mean values, and one `Logger` object for the raw values for each context. The `accumulate_means` context cannot be called inside an already open `accumulate_means` context. The `imitation.util.logger.configure` function can be used to easily construct a `HierarchicalLogger` object.

`imitation.util.networks`: This module provides some additional neural network layers that can be used for imitation like `RunningNorm` and `EMANorm` that normalize their inputs. The module also provides functions like `build_mlp` and `build_cnn` to quickly build neural networks.

`imitation.util.util`: This module provides miscellaneous util functions like `make_vec_env` to easily construct vectorized environments and `safe_to_tensor` that converts a NumPy array to a PyTorch tensor.

`imitation.util.video_wrapper.VideoWrapper`: A wrapper to record rendered videos from an environment.

## 3.3 Contributing

### 3.3.1 Code of Conduct

To ensure that the imitation community remains open and inclusive, we have a few ground rules that we ask contributors to adhere to. This isn't an exhaustive list of things that you can't do. Rather, take it in the spirit in which it's intended — a guide to make it easier to enrich all of us and the technical communities in which we participate.

- **Be friendly and patient**.
- **Be welcoming**. We strive to be a community that welcomes and supports people of all backgrounds and identities. This includes, but is not limited to members of any race, ethnicity, culture, national origin, colour, immigration status, social and economic class, educational level, sex, sexual orientation, gender identity and expression, age, size, family status, political belief, religion, and mental and physical ability.
- **Be considerate**. Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and you should take those consequences into account when making decisions. Remember that we're a world-wide community, so you might not be communicating in someone else's primary language.
- **Be respectful**. Not all of us will agree all the time, but disagreement is no excuse for poor behavior and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. Members of the imitation community should be respectful when dealing with other members as well as with people outside the imitation community.
- **Be careful in the words that you choose**. We are a community of professionals, and we conduct ourselves professionally. Be kind to others. Do not insult or put down other participants. Harassment and other exclusionary behavior aren't acceptable. This includes, but is not limited to:
  - Violent threats or language directed against another person.
  - Discriminatory jokes and language.
  - Posting sexually explicit or violent material.
  - Posting (or threatening to post) other people's personally identifying information without their consent ("doxing").
  - Personal insults, especially those using racist or sexist terms.
  - Unwelcome sexual attention.
  - Advocating for, or encouraging, any of the above behavior.
  - Repeated harassment of others. In general, if someone asks you to stop, then stop.
- **When we disagree, try to understand why**. It is important that we resolve disagreements and differing views constructively. Focus on helping to resolve issues and learning from mistakes.

Adapted from the original text courtesy of the Django project, licensed under a Creative Commons Attribution 3.0 License.

### 3.3.2 Ways to contribute

There are four main ways you can contribute to imitation:

- *Reporting bugs*
- *Suggesting new features*
- *Contributing to the documentation*
- *Contributing to the codebase*

Please note that by contributing to the project, you are agreeing to license your work under *imitation's MIT license*, as per GitHub's terms of service.

#### Reporting bugs

This section guides you through submitting a new bug report for imitation. Following the guidelines below helps maintainers and the community understand your report and reproduce the issue.

You can submit a new bug report by creating an issue on GitHub and labeling it as a *bug*. **Before you do so, please make sure that**:

- You are using the latest stable version of imitation — to check your version, run `pip show imitation`,
- You have read the relevant section of the documentation that relates to your issue,
- You have checked existing bug reports to make sure that your issue has not already been reported, and
- You have a minimal, reproducible example of the issue.

When submitting a bug report, please **include the following information**:

- A clear, concise description of the bug,
- A minimal, reproducible example of the bug, with installation instructions, code, and error message,
- Information on your OS name and version, Python version, and other relevant information (e.g. hardware configuration if using the GPU), and
- Whether the problem arose when upgrading to a certain version of imitation, and if so, what version.

#### Suggesting new features

This section explains how you can submit a new feature request, including completely new features and minor improvements to existing functionality. Following these guidelines helps maintainers and the community understand your request and intended use cases and find related suggestions.

You can submit a new bug report by creating an issue on GitHub and labeling it as an *enhancement*. **Before you do so, please make sure that**:

- You have checked the documentation that relates to your request, as it may be that such feature is already available,
- You have checked existing feature requests to make sure that there is no similar request already under discussion, and
- You have a minimal use case that describes the relevance of the feature.

When you **submit the feature request**:

- Use a clear and descriptive title for the GitHub issue to easily identify the suggestion.
- Describe the current behavior, and explain what behavior you expected to see instead and why.

- If you want to request an API change, provide examples of how the feature would be used.
- If you want to request a new algorithm implementation, please provide a link to the relevant paper or publication.

### Contributing to the documentation

One of the simplest ways to start contributing to imitation is through improving the documentation. Currently, our documentation has some gaps, and we would love to have you help us fill them. You can help by adding missing sections of the API docs, editing existing content to make it more readable, clear and accessible, or contributing new content, such as tutorials and FAQs.

If you have struggled to understand something about our codebase and managed to figure it out in the end, please consider improving the relevant documentation section, or adding a tutorial or a FAQ entry, so that other users can learn from your experience.

Before submitting a pull request, please create an issue with the *documentation* label so that we can track the gap. You can then reference the issue in your pull request by including the issue number.

### Contributing to the codebase

You can contribute to the codebase by proposing solutions to issues or feature suggestions you've raised yourself, or selecting an existing issue to work on. Please, make sure to create an issue on GitHub before you start working on a pull request, as explained in *Reporting bugs* and *Suggesting new features*.

Once you're ready to start working on your pull request, please make sure to follow our **coding style guidelines**:

- PEP8, with line width 88.
- Use the `black` autoformatter.
- Follow the Google Python Style Guide unless it conflicts with the above. Examples of Google-style docstrings can be found here.

**Before you submit**, please make sure that:

- Your PR includes unit tests for any new features.
- Your PR includes type annotations, except when it would make the code significantly more complex.
- You have run the unit tests and there are no errors. We use `pytest` for unit testing: run `pytest tests/` to run the test suite.
- You should run `pre-commit run` to run linting and static type checks. We use `pytype` for static type analysis.

You may wish to configure this as a Git commit hook:

```
pre-commit install
```

These checks are run on CircleCI and are required to pass before merging. Additionally, we track test coverage by CodeCov and require that code coverage should not decrease. This can be overridden by maintainers in exceptional cases. Files in `imitation/{examples,scripts}/` have no coverage requirements.

Thank you for your interest in imitation!

As an open-source project, we welcome contributions from all users, and are always open to any feedback or suggestions. This section of the documentation is intended to help you understand the process of contributing to the project.

To keep the community open and inclusive, we have developed a *Code of Conduct*. If you are not familiar with our Code of Conduct, take a minute to read it before starting your first contribution.

## 3.4 Release Notes

### 3.4.1 v1.0.0 – first stable release

*Released on 2023-10-31 - GitHub - PyPI*

### 3.4.2 v0.4.0

*Released on 2023-07-17 - GitHub - PyPI*

### 3.4.3 v0.3.1

*Released on 2022-07-29 - GitHub - PyPI*

### 3.4.4 v0.3.0: Major improvements

*Released on 2022-07-26 - GitHub - PyPI*

### 3.4.5 v0.2.0: First PyTorch release

*Released on 2020-10-23 - GitHub - PyPI*

### 3.4.6 v0.1.1: Final TF1 release

*Released on 2020-09-01 - GitHub - PyPI*

### 3.4.7 v0.1.0: Initial release

*Released on 2020-05-09 - GitHub - PyPI*

## 3.5 License

This license is also available on the project repository.

MIT License

Copyright (c) 2019-2022 Center for Human-Compatible AI and Google LLC

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT

HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## 3.6 Index

- genindex
- modindex

# PYTHON MODULE INDEX