seals

Center for Human-Compatible AI

Oct 28, 2022

USER GUIDE

The Suite of Environments for Algorithms that Learn Specifications, or *seals*, is a toolkit for evaluating specification learning algorithms, such as reward or imitation learning. The environments are compatible with Gym, but are designed to test algorithms that learn from user data, without requiring a procedurally specified reward function.

There are two types of environments in *seals*:

- **Diagnostic Tasks** which test individual facets of algorithm performance in isolation.
- **Renovated Environments**, adaptations of widely-used benchmarks such as MuJoCo continuous control tasks to be suitable for specification learning benchmarks. In particular, this involves removing any side-channel sources of reward information (such as episode boundaries, the score appearing in the observation, etc) and including all the information needed to compute the reward in the observation space.

seals is under active development and we intend to add more categories of tasks soon.

CHAPTER

ONE

INSTALLATION INSTRUCTIONS

To install the latest release from PyPi, run:

```
pip install seals
```

We make releases periodically, but if you wish to use the latest version of the code, you can always install directly from Git master:

```
pip install git+https://github.com/HumanCompatibleAI/seals.git
```

seals has optional dependencies needed by some subset of environments. In particular, to use MuJoCo environments, you will need to install MuJoCo 1.5 and then run:

```
pip install seals[mujoco]
```

You may need to install some other binary dependencies: see the instructions in Gym and mujoco-py for further information.

You can also use our Docker image which includes all necessary binary dependencies. You can either build it from the `Dockerfile`, or by downloading a pre-built image:

```
docker pull humancompatibleai/seals:base
```

CHAPTER

TWO

DIAGNOSTIC TASKS

Diagnostic tasks test individual facets of algorithm performance in isolation.

2.1 Branching

Gym ID: `seals/Branching-v0`

2.2 EarlyTerm

Gym ID: `seals/EarlyTermPos-v0` and `seals/EarlyTermNeg-v0`

2.3 InitShift

Gym ID: `seals/InitShiftTrain-v0` and `seals/InitShiftTest-v0seals/EarlyTermPos-v0`

2.4 LargestSum

Gym ID: `seals/LargestSum-v0`

2.5 NoisyObs

Gym ID: `seals/NoisyObs-v0`

2.6 Parabola

Gym ID: `seals/Parabola-v0`

2.7 ProcGoal

Gym ID: `seals/ProcGoal-v0`

2.8 RiskyPath

Gym ID: `seals/RiskyPath-v0`

2.9 Sort

Gym ID: `seals/Sort-v0`

CHAPTER

THREE

RENOVATED ENVIRONMENTS

These environments are adaptations of widely-used reinforcement learning benchmarks from Gym, modified to be suitable for benchmarking specification learning algorithms. In particular, we:

- Make episodes fixed length. Since episode termination conditions are often correlated with reward, variable-length episodes provide a side-channel of reward information that algorithms can exploit. Critically, episode boundaries do not exist outside of simulation: in the real-world, a human must often "reset" the RL algorithm.

  Moreover, many algorithms do not properly handle episode termination, and so are biased towards shorter or longer episode boundaries. This confounds evaluation, making some algorithms appear spuriously good or bad depending on if their bias aligns with the task objective.

  For most tasks, we make the episode fixed length simply by removing the early termination condition. In some environments, such as *MountainCar*, it does not make sense to continue after the terminal state: in this case, we make the terminal state an absorbing state that is repeated until the end of the episode.

- Ensure observations include all information necessary to compute the ground-truth reward function. For some environments, this has required augmenting the observation space. We make this modification to make RL and specification learning of comparable difficulty in these environments. While in general both RL and specification learning may need to operate in partially observable environments, the observations in these relatively simple environments were typically engineered to *make RL easy*: for a fair comparison, we must therefore also provide reward learning algorithms with sufficient features to recover the reward.

In the future, we intend to add Atari tasks with the score masked, another reward side-channel.

3.1 Classic Control

3.1.1 CartPole

Gym ID: `seals/CartPole-v0`

3.1.2 MountainCar

Gym ID: `seals/MountainCar-v0`

3.2 MuJoCo

3.2.1 Ant

Gym ID: `seals/Ant-v0`

3.2.2 HalfCheetah

Gym ID: `seals/HalfCheetah-v0`

3.2.3 Hopper

Gym ID: `seals/Hopper-v0`

3.2.4 Humanoid

Gym ID: `seals/Humanoid-v0`

3.2.5 Swimmer

Gym ID: `seals/Swimmer-v0`

3.2.6 Walker2d

Gym ID: `seals/Walker2d-v0`

CHAPTER

FOUR

BASE ENVIRONMENTS

CHAPTER

FIVE

UTILITIES

Miscellaneous utilities.

class seals.util.**AbsorbAfterDoneWrapper**(*env*, *absorb_reward=0.0*, *absorb_obs=None*)

Bases: Wrapper

Transition into absorbing state instead of episode termination.

When the environment being wrapped returns *done=True*, we return an absorbing observation. This wrapper always returns *done=False*.

A convenient way to add absorbing states to environments like MountainCar.

__init__(*env*, *absorb_reward=0.0*, *absorb_obs=None*)

Initialize AbsorbAfterDoneWrapper.

Parameters

- **env** – The wrapped Env.
- **absorb_reward** – The reward returned at the absorb state.
- **absorb_obs** – The observation returned at the absorb state. If None, then repeat the final observation before absorb.

reset(**args*, ***kwargs*)

Reset the environment.

step(*action*)

Advance the environment by one step.

This wrapped *step()* always returns done=False.

After the first done is returned by the underlying Env, we enter an artificial absorb state.

In this artificial absorb state, we stop calling *self.env.step(action)* (i.e. the *action* argument is entirely ignored) and we return fixed values for obs, rew, done, and info. The values of *obs* and *rew* depend on initialization arguments. *info* is always an empty dictionary.

class seals.util.**AutoResetWrapper**(*env*)

Bases: Wrapper

Hides done=True and auto-resets at the end of each episode.

step(*action*)

When done=True, returns done=False instead and automatically resets.

When an automatic reset happens, the observation from reset is returned, and the overridden observation is stored in *info["terminal_observation"]*.

class seals.util.**ObsCastWrapper**(*env*, *dtype*)

Bases: Wrapper

Cast observations to specified dtype.

Some external environments return observations of a different type than the declared observation space. Where possible, this should be fixed upstream, but casting can be a viable workaround – especially when the returned observations are higher resolution than the observation space.

__init__(*env*, *dtype*)

Builds ObsCastWrapper.

Parameters

- **env** – the environment to wrap.
- **dtype** – the dtype to cast observations to.

reset()

Returns reset observation, cast to self.dtype.

step(*action*)

Returns (obs, rew, done, info) with obs cast to self.dtype.

seals.util.**get_gym_max_episode_steps**(*env_name*)

Get the *max_episode_steps* attribute associated with a gym Spec.

Return type
Optional[int]

seals.util.**grid_transition_fn**(*state*, *action*, *x_bounds=(-inf, inf)*, *y_bounds=(-inf, inf)*)

Returns transition of a deterministic gridworld.

Agent is bounded in the region limited by x_bounds and y_bounds, ends inclusive.

(0, 0) is interpreted to be top-left corner.

Actions: 0: Right 1: Down 2: Left 3: Up 4: Stay put

seals.util.**make_env_no_wrappers**(*env_name*, ***kwargs*)

Gym sometimes wraps envs in TimeLimit before returning from gym.make().

This helper method builds directly from spec to avoid this wrapper.

Return type
Env

seals.util.**one_hot_encoding**(*pos*, *size*)

Returns a 1-D hot encoding of a given position and size.

Return type
ndarray

seals.util.**sample_distribution**(*p*, *random*)

Samples an integer with probabilities given by p.

Return type
int

CHAPTER

SIX

HELPERS FOR UNIT-TESTING ENVIRONMENTS

CHAPTER

SEVEN

CITING SEALS

To cite this project in publications:

```
@misc{seals,
  author = {Adam Gleave and Pedro Freire and Steven Wang and Sam Toyer},
  title = {{seals}: Suite of Environments for Algorithms that Learn Specifications},
  year = {2020},
  publisher = {GitHub},
  journal = {GitHub repository},
  howpublished = {\url{https://github.com/HumanCompatibleAI/seals}},
}
```

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

G

M

O

R

S