Pokemon battle RL environment Documentation

Release 1

Sep 03, 2019

Contents:

This repository contains a Reinforcement Learning environment for Pokémon battles.

In particular, the environment consists of three parts:

- A Gym Env which serves as interface between RL agents and battle simulators
- A BattleSimulator base class, which handles typical Pokémon game state
- Simulator classes derived from BattleSimulator, which access and interact with different simulators to extract data

Currently, only a Pokemon Showdown integration is planned, but in theory this structure allows for integrations with different simulators (eg Console emulators).

CHAPTER 1

pokebattle_rl_env

1.1 pokebattle_rl_env package

1.1.1 Submodules

1.1.2 pokebattle_rl_env.battle_simulator module

1.1.3 pokebattle_rl_env.game_state module

1.1.4 pokebattle_rl_env.poke_data_queries module

1.1.5 pokebattle_rl_env.pokebattle_env module

class pokebattle_rl_env.pokebattle_env.**PokeBattleEnv**(*simulator=<pokebattle_rl_env.showdown_simulator.Sho object>*)

Bases: gym.core.Env

The Pokemon battle Reinforecement Learning environment.

A subclass of gym.core.Env, which is compatible with most Reinforcement Learning frameworks. *PokeBattleEnv* uses a pokebattle_rl_env.battle_simulator.BattleSimulator to simulate the battles.

simulator
The simulator to run battles in. Uses *pokebattle_rl_env.showdown_simulator.ShowdownSimulator* by default.

Type pokebattle_rl_env.battle_simulator.BattleSimulator

close()
Override close in your subclass to perform any necessary cleanup.

Environments will automatically close() themselves when garbage collected or when the program exits.

render(*mode='human'*)
Renders the environment.

The set of supported modes varies per environment. (And some environments do not support rendering at all.) By convention, if mode is:

- human: render to the current display or terminal and return nothing. Usually for human consumption.
- rgb_array: Return an numpy.ndarray with shape (x, y, 3), representing RGB values for an x-by-y pixel image, suitable for turning into a video.
- ansi: Return a string (str) or StringIO.StringIO containing a terminal-style text representation. The text can include newlines and ANSI escape sequences (e.g. for colors).

Note:

Make sure that your class's metadata 'render.modes' key includes the list of supported modes. It's recommended to call super() in implementations to use the functionality of this method.

Parameters **mode** (*str*) – the mode to render with

Example:

class MyEnv(Env): metadata = {'render.modes': ['human', 'rgb_array']}

def render(self, mode='human'):

if mode == 'rgb_array': return np.array(...) # return RGB frame suitable for video

elif mode == 'human': ... # pop up a window and render

else: super(MyEnv, self).render(mode=mode) # just raise an exception

reset()
Resets the state of the environment and returns an initial observation.

Returns the initial observation.

Return type observation (object)

seed(*seed=None*)
Sets the seed for this env's random number generator(s).

Note: Some environments use multiple pseudorandom number generators. We want to capture all such seeds used in order to ensure that there aren't accidental correlations between multiple generators.

Returns

Returns the list of seeds used in this env's random number generators. The first value in the list should be the "main" seed, or the value which a reproducer should pass to 'seed'. Often, the main seed equals the provided 'seed', but this won't be true if seed=None, for example.

Return type list<bigint>

step(*action*)
Run one timestep of the environment's dynamics. When end of episode is reached, you are responsible for calling *reset()* to reset this environment's state.

Accepts an action and returns a tuple (observation, reward, done, info).

Parameters **action** (*object*) – an action provided by the agent

Returns agent's observation of the current environment reward (float) : amount of reward returned after previous action done (bool): whether the episode has ended, in which case further step() calls will return undefined results info (dict): contains auxiliary diagnostic information (helpful for debugging, and sometimes learning)

Return type observation (object)

1.1.6 pokebattle_rl_env.showdown_simulator module

class pokebattle_rl_env.showdown_simulator.**ShowdownConnection**(*ws_host*, *ws_port*, *ws_ssl*, *web_host*, *web_port*, *web_ssl*)

Bases: object

Holds information on how to connect to various endpoints of a specific Pokemon Showdown instance.

There are two useful endpoints of each Pokemon Showdown instance:

- The WebSocket endpoint, which enables user interaction and is used to run battles
- The HTTP endpoint, which displays the client and is used to view battles

DEFAULT_PUBLIC_CONNECTION uses the default connection for the public instance at https://play.pokemonshowdown.com. DEFAULT_LOCAL_CONNECTION uses the default connection for the local instance at https://localhost:8000. Specify a new instance of this class to use a custom Pokemon Showdown instance not hosted locally.

ws_host
The hostname of the WebSocket endpoint. Can be different from *web_host*.

Type str

ws_port
The port of the WebSocket endpoint.

Type int

ws_ssl
Whether to use the WebSocket Secure protocol. Keep in mind to use the corresponding *ws_port* (most likely 433).

Type bool

web_host
The hostname of the HTTP endpoint. Can be different from *ws_host*.

Type str

web_port
The port of the HTTP endpoint.

Type int

web_ssl
Whether to use HTTPS. Keep in mind to use the corresponding *web_port* (most likely 433).

Type bool

class pokebattle_rl_env.showdown_simulator.**ShowdownSimulator**(*auth=''*, *self_play=False*, *connection=<pokebattle_rl_env.showdown_simulator.ShowdownConnection object>*, *logging_file=None*)

Bases: pokebattle_rl_env.battle_simulator.BattleSimulator

A pokebattle_rl_env.battle_simulator.BattleSimulator using Pokemon Showdown as backend.

View ongoing battles at https://play.pokemonshowdown.com/*room_id* if local is False or at http://localhost:8000/*room_id* if otherwise.

state
The current state of the battle.

Type pokebattle_rl_env.game_state.GameState

auth
The authentication method to use to log into https://pokemonshowdown.com. Options:

- empty string: Log into a temporary account.
- *'register'*: Generate a username and password to register an account. The credentials will be output on the console.
- path to authentication file: Logs into an account specified in a text file, where the first line specifies the username and the second line specifies the password.

Type str

self_play
Whether to use self play. Note that this is a naive self play-implementation. In fact, agents simply play against other agents - a temporary text file keeps track of the battles. Thus, self play only works if *number of agents % 2 == 0*. If *self_play* is false, the agent will battle against random human opponents. Keep in mind that this self-play implementation is redundant if multiple agents are deployed on a local Pokemon Showdown instance (see *connection*) without human players. If https://github.com/Zarel/Pokemon-Showdown/blob/master/ladders.js#L470 and https://github.com/Zarel/Pokemon-Showdown/blob/master/ladders.js#L470 is removed, they will battle against each other automatically.

Type bool

connection
Details which Pokemon Showdown connection to use. The default connection is to the local instance at https://localhost:8000. Use a local instance of Pokemon Showdown whenever possible. See https://github.com/Zarel/Pokemon-Showdown for installation instructions. Obviously, if self play is not desired, using a local/custom instance is only recommended if there are human players on it. Otherwise, set *connection* to DEFAULT_PUBLIC_CONNECTION to use the public connection at https://play.pokemonshowdown.com.

Type *pokebattle_rl_env.showdown_simulator.ShowdownConnection*

logging_file
Specify the path to a file to log debug output.

Type bool

room_id
The string used to identify the current battle (room).

Type str

close()
Closes the connection to the WebSocket endpoint.

render(*mode='human'*)
Renders the ongoing battle, if there is any.

Parameters **mode** (*str*) – Details the rendering mode. Currently, only mode *human* is supported. *human* will simply open the ongoing battle in a web browser (if one exists). Therefore, it is advised to call *render()* only once per battle.

reset()
Resets the simulator to its initial state. Call this function prior to calling `act()`. It automatically sets up a new battle, even if there exists an ongoing battle.

pokebattle_rl_env.showdown_simulator.**auth_temp_user**(*challstr*, *username*)
Logs into a temporary user account on https://pokemonshowdown.com. The account is not password protected and deleted after a day.

Parameters

- **challstr** (*str*) – The challenge string sent by the Pokemon Showdown server. Obtain this string by connecting to the Pokemon Showdown WebSocket.
- **username** (*str*) – The username to register.

Returns The assertion string used as authentication with the WebSocket.

Return type str

Raises `ValueError` – If at least one of the parameters is empty.

pokebattle_rl_env.showdown_simulator.**ident_to_name**(*ident*)
Retrieves the pokemon name out of a pokemon identification string.

Parameters **ident** (*str*) – The pokemon identification string.

Returns The name of the pokemon

Return type str

Examples

```
>>> ident_to_name('p1a: Metagross')
'Metagross'
```

pokebattle_rl_env.showdown_simulator.**ident_to_pokemon**(*ident*, *state*, *opponent_short=None*)

pokebattle_rl_env.showdown_simulator.**login**(*challstr*, *username*, *password*)
Logs into an existing account on https://pokemonshowdown.com.

Parameters

- **challstr** (*str*) – The challenge string sent by the Pokemon Showdown server. Obtain this string by connecting to the Pokemon Showdown WebSocket.
- **username** (*str*) – The username to login.
- **password** (*str*) – The password to login.

Returns The assertion string used as authentication with the WebSocket.

Return type str

Raises `ValueError` – If at least one of the parameters is empty or the authentication using the provided credentials failed.

`pokebattle_rl_env.showdown_simulator.`**`random`**`()` → x in the interval [0, 1).

`pokebattle_rl_env.showdown_simulator.`**`register`**(*challstr*, *username*, *password*)
Registers an account on https://pokemonshowdown.com.

Parameters

- **`challstr`** (*`str`*) – The challenge string sent by the Pokemon Showdown server. Obtain this string by connecting to the Pokemon Showdown WebSocket.
- **`username`** (*`str`*) – The username to register. Must be unique and not yet chosen.
- **`password`** (*`str`*) – The password to register. Must be unique and not yet chosen.

Returns The assertion string used as authentication with the WebSocket.

Return type str

Raises `ValueError` – If at least one of the parameters is empty or the authentication using the provided credentials failed.

1.1.7 pokebattle_rl_env.util module

1.1.8 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index