Stable Baselines3 Documentation

Release 1.0

Stable Baselines3 Contributors

Mar 12, 2023

USER GUIDE

Contrib package for Stable Baselines3 (SB3) - Experimental code.

Github repository: https://github.com/Stable-Baselines-Team/stable-baselines3-contrib

SB3 repository: https://github.com/DLR-RM/stable-baselines3

RL Baselines3 Zoo (collection of pre-trained agents): https://github.com/DLR-RM/rl-baselines3-zoo

RL Baselines3 Zoo also offers a simple interface to train, evaluate agents and do hyperparameter tuning.

CHAPTER

ONE

INSTALLATION

1.1 Prerequisites

Please read Stable-Baselines3 installation guide first.

1.1.1 Stable Release

To install Stable Baselines3 contrib with pip, execute:

```
pip install sb3-contrib
```

1.2 Bleeding-edge version

```
pip install git+https://github.com/Stable-Baselines-Team/stable-baselines3-contrib/
```

1.3 Development version

To contribute to Stable-Baselines3, with support for running tests and building the documentation.

```
git clone https://github.com/Stable-Baselines-Team/stable-baselines3-contrib/ && cd␣
↪stable-baselines3-contrib
pip install -e .
```

CHAPTER

TWO

RL ALGORITHMS

This table displays the rl algorithms that are implemented in the Stable Baselines3 contrib project, along with some useful characteristics: support for discrete/continuous actions, multiprocessing.

Name	`Box`	`Discrete`	`MultiDiscrete`	`MultiBinary`	Multi Processing
TQC	✓				
QR-DQN		✓			

Note: Non-array spaces such as `Dict` or `Tuple` are not currently supported by any algorithm.

Actions `gym.spaces`:

- `Box`: A N-dimensional box that contains every point in the action space.
- `Discrete`: A list of possible actions, where each timestep only one of the actions can be used.
- `MultiDiscrete`: A list of possible actions, where each timestep only one action of each discrete set can be used.
- `MultiBinary`: A list of possible actions, where each timestep any of the actions can be used in any combination.

CHAPTER

THREE

EXAMPLES

3.1 TQC

Train a Truncated Quantile Critics (TQC) agent on the Pendulum environment.

```
from sb3_contrib import TQC

model = TQC("MlpPolicy", "Pendulum-v0", top_quantiles_to_drop_per_net=2, verbose=1)
model.learn(total_timesteps=10000, log_interval=4)
model.save("tqc_pendulum")
```

3.2 QR-DQN

Train a Quantile Regression DQN (QR-DQN) agent on the CartPole environment.

```
from sb3_contrib import QRDQN

policy_kwargs = dict(n_quantiles=50)
model = QRDQN("MlpPolicy", "CartPole-v1", policy_kwargs=policy_kwargs, verbose=1)
model.learn(total_timesteps=10000, log_interval=4)
model.save("qrdqn_cartpole")
```

CHAPTER

FOUR

TQC

Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics (TQC). Truncated Quantile Critics (TQC) builds on SAC, TD3 and QR-DQN, making use of quantile regression to predict a distribution for the value function (instead of a mean value). It truncates the quantiles predicted by different networks (a bit as it is done in TD3).

Available Policies

4.1 Notes

- Original paper: https://arxiv.org/abs/2005.04269
- Original Implementation: https://github.com/bayesgroup/tqc_pytorch

4.2 Can I use?

- Recurrent policies:
- Multi processing:
- Gym spaces:

Space	Action	Observation
Discrete		✓
Box	✓	✓
MultiDiscrete		✓
MultiBinary		✓

4.3 Example

```
import gym
import numpy as np

from sb3_contrib import TQC

env = gym.make("Pendulum-v0")

policy_kwargs = dict(n_critics=2, n_quantiles=25)
model = TQC("MlpPolicy", env, top_quantiles_to_drop_per_net=2, verbose=1, policy_
↪kwargs=policy_kwargs)
model.learn(total_timesteps=10000, log_interval=4)
model.save("tqc_pendulum")

del model # remove to demonstrate saving and loading

model = TQC.load("tqc_pendulum")

obs = env.reset()
while True:
    action, _states = model.predict(obs, deterministic=True)
    obs, reward, done, info = env.step(action)
    env.render()
    if done:
      obs = env.reset()
```

4.4 Results

Result on the PyBullet benchmark (1M steps) and on BipedalWalkerHardcore-v3 (2M steps) using 3 seeds. The complete learning curves are available in the associated PR.

The main difference with SAC is on harder environments (BipedalWalkerHardcore, Walker2D).

Note: Hyperparameters from the gSDE paper were used (as they are tuned for SAC on PyBullet envs), including using gSDE for the exploration and not the unstructured Gaussian noise but this should not affect results in simulation.

Note: We are using the open source PyBullet environments and not the MuJoCo simulator (as done in the original paper). You can find a complete benchmark on PyBullet envs in the gSDE paper if you want to compare TQC results to those of A2C/PPO/SAC/TD3.

Environments	SAC	TQC
	gSDE	gSDE
HalfCheetah	2984 +/- 202	3041 +/- 157
Ant	3102 +/- 37	3700 +/- 37
Hopper	2262 +/- 1	2401 +/- 62*
Walker2D	2136 +/- 67	2535 +/- 94
BipedalWalkerHardcore	13 +/- 18	228 +/- 18

* with tuned hyperparameter `top_quantiles_to_drop_per_net` taken from the original paper

4.4.1 How to replicate the results?

Clone RL-Zoo and checkout the branch `feat/tqc`:

```
git clone https://github.com/DLR-RM/rl-baselines3-zoo
cd rl-baselines3-zoo/
git checkout feat/tqc
```

Run the benchmark (replace `$ENV_ID` by the envs mentioned above):

```
python train.py --algo tqc --env $ENV_ID --eval-episodes 10 --eval-freq 10000
```

Plot the results:

```
python scripts/all_plots.py -a tqc -e HalfCheetah Ant Hopper Walker2D␣
↪BipedalWalkerHardcore -f logs/ -o logs/tqc_results
python scripts/plot_from_file.py -i logs/tqc_results.pkl -latex -l TQC
```

4.5 Comments

This implementation is based on SB3 SAC implementation and uses the code from the original TQC implementation for the quantile huber loss.

4.6 Parameters

4.7 TQC Policies

CHAPTER

FIVE

QR-DQN

Quantile Regression DQN (QR-DQN) builds on Deep Q-Network (DQN) and make use of quantile regression to explicitly model the distribution over returns, instead of predicting the mean return (DQN).

Available Policies

MlpPolicy	alias of `sb3_contrib.qrdqn.policies.QRDQNPolicy`
CnnPolicy	Policy class for QR-DQN when using images as input.

5.1 Notes

- Original paper: https://arxiv.org/abs/1710.100442
- Distributional RL (C51): https://arxiv.org/abs/1707.06887
- Further reference: https://github.com/amy12xx/ml_notes_and_reports/blob/master/distributional_rl/QRDQN.pdf

5.2 Can I use?

- Recurrent policies:
- Multi processing:
- Gym spaces:

Space	Action	Observation
Discrete	✓	✓
Box		✓
MultiDiscrete		✓
MultiBinary		✓

5.3 Example

```
import gym

from sb3_contrib import QRDQN

env = gym.make("CartPole-v1")

policy_kwargs = dict(n_quantiles=50)
model = QRDQN("MlpPolicy", env, policy_kwargs=policy_kwargs, verbose=1)
model.learn(total_timesteps=10000, log_interval=4)
model.save("qrdqn_cartpole")

del model # remove to demonstrate saving and loading

model = QRDQN.load("qrdqn_cartpole")

obs = env.reset()
while True:
    action, _states = model.predict(obs, deterministic=True)
    obs, reward, done, info = env.step(action)
    env.render()
    if done:
      obs = env.reset()
```

5.4 Results

Result on Atari environments (10M steps, Pong and Breakout) and classic control tasks using 3 and 5 seeds.

The complete learning curves are available in the associated PR.

Note: QR-DQN implementation was validated against Intel Coach one which roughly compare to the original paper results (we trained the agent with a smaller budget).

Environments	QR-DQN	DQN
Breakout	413 +/- 21	~300
Pong	20 +/- 0	~20
CartPole	386 +/- 64	500 +/- 0
MountainCar	-111 +/- 4	-107 +/- 4
LunarLander	168 +/- 39	195 +/- 28
Acrobot	-73 +/- 2	-74 +/- 2

5.4.1 How to replicate the results?

Clone RL-Zoo fork and checkout the branch `feat/qrdqn`:

```
git clone https://github.com/ku2482/rl-baselines3-zoo/
cd rl-baselines3-zoo/
git checkout feat/qrdqn
```

Run the benchmark (replace `$ENV_ID` by the envs mentioned above):

```
python train.py --algo qrdqn --env $ENV_ID --eval-episodes 10 --eval-freq 10000
```

Plot the results:

```
python scripts/all_plots.py -a qrdqn -e Breakout Pong -f logs/ -o logs/qrdqn_results
python scripts/plot_from_file.py -i logs/qrdqn_results.pkl -latex -l QR-DQN
```

5.5 Parameters

class sb3_contrib.qrdqn.**QRDQN**(*policy*, *env*, *learning_rate=5e-05*, *buffer_size=1000000*, *learning_starts=50000*, *batch_size=32*, *tau=1.0*, *gamma=0.99*, *train_freq=4*, *gradient_steps=1*, *optimize_memory_usage=False*, *target_update_interval=10000*, *exploration_fraction=0.005*, *exploration_initial_eps=1.0*, *exploration_final_eps=0.01*, *max_grad_norm=None*, *tensorboard_log=None*, *create_eval_env=False*, *policy_kwargs=None*, *verbose=0*, *seed=None*, *device='auto'*, *_init_setup_model=True*)

Quantile Regression Deep Q-Network (QR-DQN) Paper: https://arxiv.org/abs/1710.10044 Default hyperparameters are taken from the paper and are tuned for Atari games.

Parameters

- **policy** (`Union[str, Type[QRDQNPolicy]]`) – The policy model to use (MlpPolicy, CnnPolicy, ...)
- **env** (`Union[Env, VecEnv, str]`) – The environment to learn from (if registered in Gym, can be str)
- **learning_rate** (`Union[float, Callable[[float], float]]`) – The learning rate, it can be a function of the current progress remaining (from 1 to 0)
- **buffer_size** (`int`) – size of the replay buffer
- **learning_starts** (`int`) – how many steps of the model to collect transitions for before learning starts
- **batch_size** (`Optional[int]`) – Minibatch size for each gradient update
- **tau** (`float`) – the soft update coefficient ("Polyak update", between 0 and 1) default 1 for hard update
- **gamma** (`float`) – the discount factor
- **train_freq** (`int`) – Update the model every `train_freq` steps. Alternatively pass a tuple of frequency and unit like `(5, "step")` or `(2, "episode")`.

- **gradient_steps** (int) – How many gradient steps to do after each rollout (see train_freq and n_episodes_rollout) Set to -1 means to do as many gradient steps as steps done in the environment during the rollout.
- **optimize_memory_usage** (bool) – Enable a memory efficient variant of the replay buffer at a cost of more complexity. See https://github.com/DLR-RM/stable-baselines3/issues/37#issuecomment-637501195
- **target_update_interval** (int) – update the target network every target_update_interval environment steps.
- **exploration_fraction** (float) – fraction of entire training period over which the exploration rate is reduced
- **exploration_initial_eps** (float) – initial value of random action probability
- **exploration_final_eps** (float) – final value of random action probability
- **max_grad_norm** (Optional[float]) – The maximum value for the gradient clipping (if None, no clipping)
- **tensorboard_log** (Optional[str]) – the log location for tensorboard (if None, no logging)
- **create_eval_env** (bool) – Whether to create a second environment that will be used for evaluating the agent periodically. (Only available when passing string for the environment)
- **policy_kwargs** (Optional[Dict[str, Any]]) – additional arguments to be passed to the policy on creation
- **verbose** (int) – the verbosity level: 0 no output, 1 info, 2 debug
- **seed** (Optional[int]) – Seed for the pseudo random generators
- **device** (Union[device, str]) – Device (cpu, cuda, ...) on which the code should be run. Setting it to auto, the code will be run on the GPU if possible.
- **_init_setup_model** (bool) – Whether or not to build the network at the creation of the instance

collect_rollouts(*env*, *callback*, *train_freq*, *replay_buffer*, *action_noise=None*, *learning_starts=0*, *log_interval=None*)

Collect experiences and store them into a ReplayBuffer.

Parameters

- **env** (VecEnv) – The training environment
- **callback** (BaseCallback) – Callback that will be called at each step (and at the beginning and end of the rollout)
- **train_freq** (TrainFreq) – How much experience to collect by doing rollouts of current policy. Either TrainFreq(<n>, TrainFrequencyUnit.STEP) or TrainFreq(<n>, TrainFrequencyUnit.EPISODE) with <n> being an integer greater than 0.
- **action_noise** (Optional[ActionNoise]) – Action noise that will be used for exploration Required for deterministic policy (e.g. TD3). This can also be used in addition to the stochastic policy for SAC.
- **learning_starts** (int) – Number of steps before learning for the warm-up phase.
- **replay_buffer** (ReplayBuffer) –
- **log_interval** (Optional[int]) – Log data every log_interval episodes

Return type RolloutReturn

Returns

get_env()
Returns the current environment (can be None if not defined).

Return type Optional[VecEnv]

Returns The current environment

get_parameters()
Return the parameters of the agent. This includes parameters from different networks, e.g. critics (value functions) and policies (pi functions).

Return type Dict[str, Dict]

Returns Mapping of from names of the objects to PyTorch state-dicts.

get_vec_normalize_env()
Return the VecNormalize wrapper of the training env if it exists.

Return type Optional[VecNormalize]

Returns The VecNormalize env.

learn(*total_timesteps*, *callback=None*, *log_interval=4*, *eval_env=None*, *eval_freq=- 1*, *n_eval_episodes=5*, *tb_log_name='QRDQN'*, *eval_log_path=None*, *reset_num_timesteps=True*)
Return a trained model.

Parameters

- **total_timesteps** (int) – The total number of samples (env steps) to train on
- **callback** (Union[None, Callable, List[BaseCallback], BaseCallback]) – callback(s) called at every step with state of the algorithm.
- **log_interval** (int) – The number of timesteps before logging.
- **tb_log_name** (str) – the name of the run for TensorBoard logging
- **eval_env** (Union[Env, VecEnv, None]) – Environment that will be used to evaluate the agent
- **eval_freq** (int) – Evaluate the agent every eval_freq timesteps (this may vary a little)
- **n_eval_episodes** (int) – Number of episode to evaluate the agent
- **eval_log_path** (Optional[str]) – Path to a folder where the evaluations will be saved
- **reset_num_timesteps** (bool) – whether or not to reset the current timestep number (used in logging)

Return type OffPolicyAlgorithm

Returns the trained model

classmethod load(*path*, *env=None*, *device='auto'*, *custom_objects=None*, *print_system_info=False*, *force_reset=True*, ***kwargs*)
Load the model from a zip-file

Parameters

- **path** (Union[str, Path, BufferedIOBase]) – path to the file (or a file-like) where to load the agent from

- **env** (Union[Env, VecEnv, None]) – the new environment to run the loaded model on (can be None if you only need prediction from a trained model) has priority over any saved environment
- **device** (Union[device, str]) – Device on which the code should run.
- **custom_objects** (Optional[Dict[str, Any]]) – Dictionary of objects to replace upon loading. If a variable is present in this dictionary as a key, it will not be deserialized and the corresponding item will be used instead. Similar to custom_objects in keras.models.load_model. Useful when you have an object in file that can not be deserialized.
- **print_system_info** (bool) – Whether to print system info from the saved model and the current system info (useful to debug loading issues)
- **force_reset** (bool) – Force call to reset() before training to avoid unexpected behavior. See https://github.com/DLR-RM/stable-baselines3/issues/597
- **kwargs** – extra arguments to change the model when loading

Return type BaseAlgorithm

load_replay_buffer(*path*, *truncate_last_traj=True*)
Load a replay buffer from a pickle file.

Parameters

- **path** (Union[str, Path, BufferedIOBase]) – Path to the pickled replay buffer.
- **truncate_last_traj** (bool) – When using HerReplayBuffer with online sampling: If set to True, we assume that the last trajectory in the replay buffer was finished (and truncate it). If set to False, we assume that we continue the same trajectory (same episode).

Return type None

property logger: stable_baselines3.common.logger.Logger
Getter for the logger object.

Return type Logger

predict(*observation*, *state=None*, *mask=None*, *deterministic=False*)
Overrides the base_class predict function to include epsilon-greedy exploration.

Parameters

- **observation** (ndarray) – the input observation
- **state** (Optional[ndarray]) – The last states (can be None, used in recurrent policies)
- **mask** (Optional[ndarray]) – The last masks (can be None, used in recurrent policies)
- **deterministic** (bool) – Whether or not to return deterministic actions.

Return type Tuple[ndarray, Optional[ndarray]]

Returns the model's action and the next state (used in recurrent policies)

save(*path*, *exclude=None*, *include=None*)
Save all the attributes of the object and the model parameters in a zip-file.

Parameters

- **path** (Union[str, Path, BufferedIOBase]) – path to the file where the rl agent should be saved
- **exclude** (Optional[Iterable[str]]) – name of parameters that should be excluded in addition to the default ones

- **include** (Optional[Iterable[str]]) – name of parameters that might be excluded but should be included anyway

Return type None

save_replay_buffer(*path*)
Save the replay buffer as a pickle file.

Parameters **path** (Union[str, Path, BufferedIOBase]) – Path to the file where the replay buffer should be saved. if path is a str or pathlib.Path, the path is automatically created if necessary.

Return type None

set_env(*env*, *force_reset=True*)
Checks the validity of the environment, and if it is coherent, set it as the current environment. Furthermore wrap any non vectorized env into a vectorized checked parameters: - observation_space - action_space

Parameters

- **env** (Union[Env, VecEnv]) – The environment for learning a policy
- **force_reset** (bool) – Force call to `reset()` before training to avoid unexpected behavior. See issue https://github.com/DLR-RM/stable-baselines3/issues/597

Return type None

set_logger(*logger*)
Setter for for logger object.

Warning: When passing a custom logger object, this will overwrite `tensorboard_log` and `verbose` settings passed to the constructor.

Return type None

set_parameters(*load_path_or_dict*, *exact_match=True*, *device='auto'*)
Load parameters from a given zip-file or a nested dictionary containing parameters for different modules (see `get_parameters`).

Parameters

- **load_path_or_iter** – Location of the saved data (path or file-like, see `save`), or a nested dictionary containing nn.Module parameters used by the policy. The dictionary maps object names to a state-dictionary returned by `torch.nn.Module.state_dict()`.
- **exact_match** (bool) – If True, the given parameters should include parameters for each module and each of their parameters, otherwise raises an Exception. If set to False, this can be used to update only specific parameters.
- **device** (Union[device, str]) – Device on which the code should run.

Return type None

set_random_seed(*seed=None*)
Set the seed of the pseudo-random generators (python, numpy, pytorch, gym, action_space)

Parameters **seed** (Optional[int]) –

Return type None

train(*gradient_steps*, *batch_size=100*)
Sample the replay buffer and do the updates (gradient descent and update target networks)

Return type None

5.6 QR-DQN Policies

sb3_contrib.qrdqn.**MlpPolicy**
alias of sb3_contrib.qrdqn.policies.QRDQNPolicy

class sb3_contrib.qrdqn.policies.**QRDQNPolicy**(*observation_space*, *action_space*, *lr_schedule*, *n_quantiles=200*, *net_arch=None*, *activation_fn=<class 'torch.nn.modules.activation.ReLU'>*, *features_extractor_class=<class 'stable_baselines3.common.torch_layers.FlattenExtractor'>*, *features_extractor_kwargs=None*, *normalize_images=True*, *optimizer_class=<class 'torch.optim.adam.Adam'>*, *optimizer_kwargs=None*)

Policy class with quantile and target networks for QR-DQN.

Parameters

- **observation_space** (Space) – Observation space
- **action_space** (Space) – Action space
- **lr_schedule** (Callable[[float], float]) – Learning rate schedule (could be constant)
- **n_quantiles** (int) – Number of quantiles
- **net_arch** (Optional[List[int]]) – The specification of the network architecture.
- **activation_fn** (Type[Module]) – Activation function
- **features_extractor_class** (Type[BaseFeaturesExtractor]) – Features extractor to use.
- **features_extractor_kwargs** (Optional[Dict[str, Any]]) – Keyword arguments to pass to the features extractor.
- **normalize_images** (bool) – Whether to normalize images or not, dividing by 255.0 (True by default)
- **optimizer_class** (Type[Optimizer]) – The optimizer to use, th.optim.Adam by default
- **optimizer_kwargs** (Optional[Dict[str, Any]]) – Additional keyword arguments, excluding the learning rate, to pass to the optimizer

forward(*obs*, *deterministic=True*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

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

Return type Tensor

class sb3_contrib.qrdqn.**CnnPolicy**(*observation_space*, *action_space*, *lr_schedule*, *n_quantiles=200*, *net_arch=None*, *activation_fn=<class 'torch.nn.modules.activation.ReLU'>*, *features_extractor_class=<class 'stable_baselines3.common.torch_layers.NatureCNN'>*, *features_extractor_kwargs=None*, *normalize_images=True*, *optimizer_class=<class 'torch.optim.adam.Adam'>*, *optimizer_kwargs=None*)

Policy class for QR-DQN when using images as input.

Parameters

- **observation_space** (Space) – Observation space
- **action_space** (Space) – Action space
- **lr_schedule** (Callable[[float], float]) – Learning rate schedule (could be constant)
- **n_quantiles** (int) – Number of quantiles
- **net_arch** (Optional[List[int]]) – The specification of the network architecture.
- **activation_fn** (Type[Module]) – Activation function
- **features_extractor_class** (Type[BaseFeaturesExtractor]) – Features extractor to use.
- **normalize_images** (bool) – Whether to normalize images or not, dividing by 255.0 (True by default)
- **optimizer_class** (Type[Optimizer]) – The optimizer to use, th.optim.Adam by default
- **optimizer_kwargs** (Optional[Dict[str, Any]]) – Additional keyword arguments, excluding the learning rate, to pass to the optimizer

CHAPTER

SIX

GYM WRAPPERS

Additional Gym Wrappers to enhance Gym environments.

6.1 TimeFeatureWrapper

class sb3_contrib.common.wrappers.**TimeFeatureWrapper**(*env*, *max_steps=1000*, *test_mode=False*)

Add remaining, normalized time to observation space for fixed length episodes. See https://arxiv.org/abs/1712.00378 and https://github.com/aravindr93/mjrl/issues/13.

Note: Only `gym.spaces.Box` and `gym.spaces.Dict` (`gym.GoalEnv`) 1D observation spaces are supported for now.

Parameters

- **env** (`Env`) – Gym env to wrap.
- **max_steps** (`int`) – Max number of steps of an episode if it is not wrapped in a `TimeLimit` object.
- **test_mode** (`bool`) – In test mode, the time feature is constant, equal to zero. This allow to check that the agent did not overfit this feature, learning a deterministic pre-defined sequence of actions.

reset()

Resets the environment with kwargs.

Return type `Union[Tuple, Dict[str, Any], ndarray, int]`

step(*action*)

Steps through the environment with action.

Return type `Tuple[Union[Tuple, Dict[str, Any], ndarray, int], float, bool, Dict]`

CHAPTER

SEVEN

CHANGELOG

7.1 Release 1.0 (2021-03-17)

7.1.1 Breaking Changes:

- Upgraded to Stable-Baselines3 >= 1.0

7.1.2 Bug Fixes:

- Fixed a bug with `QR-DQN` predict method when using `deterministic=False` with image space

7.2 Pre-Release 0.11.1 (2021-02-27)

7.2.1 Bug Fixes:

- Upgraded to Stable-Baselines3 >= 0.11.1

7.3 Pre-Release 0.11.0 (2021-02-27)

7.3.1 Breaking Changes:

- Upgraded to Stable-Baselines3 >= 0.11.0

7.3.2 New Features:

- Added `TimeFeatureWrapper` to the wrappers
- Added `QR-DQN` algorithm (@ku2482)

7.3.3 Bug Fixes:

- Fixed bug in `TQC` when saving/loading the policy only with non-default number of quantiles
- Fixed bug in `QR-DQN` when calculating the target quantiles (@ku2482, @guyk1971)

7.3.4 Deprecations:

7.3.5 Others:

- Updated `TQC` to match new SB3 version
- Updated SB3 min version
- Moved `quantile_huber_loss` to `common/utils.py` (@ku2482)

7.3.6 Documentation:

7.4 Pre-Release 0.10.0 (2020-10-28)

Truncated Quantiles Critic (TQC)

7.4.1 Breaking Changes:

7.4.2 New Features:

- Added `TQC` algorithm (@araffin)

7.4.3 Bug Fixes:

- Fixed features extractor issue (`TQC` with `CnnPolicy`)

7.4.4 Deprecations:

7.4.5 Others:

7.4.6 Documentation:

- Added initial documentation
- Added contribution guide and related PR templates

7.5 Maintainers

Stable-Baselines3 is currently maintained by Antonin Raffin (aka @araffin), Ashley Hill (aka @hill-a), Maximilian Ernestus (aka @ernestum), Adam Gleave (@AdamGleave) and Anssi Kanervisto (aka @Miffyli).

7.6 Contributors:

@ku2482 @guyk1971

CHAPTER

EIGHT

CITING STABLE BASELINES3

To cite this project in publications:

```
@misc{stable-baselines3,
  author = {Raffin, Antonin and Hill, Ashley and Ernestus, Maximilian and Gleave, Adam␣
↪and Kanervisto, Anssi and Dormann, Noah},
  title = {Stable Baselines3},
  year = {2019},
  publisher = {GitHub},
  journal = {GitHub repository},
  howpublished = {\url{https://github.com/DLR-RM/stable-baselines3}},
}
```

CHAPTER

NINE

CONTRIBUTING

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

CHAPTER

TEN

INDICES AND TABLES

- genindex
- search
- modindex

PYTHON MODULE INDEX

S

INDEX