Stable-Baselines3: Reliable Reinforcement Learning Implementations

Antonin Raffin\textsuperscript{1}  
Ashley Hill\textsuperscript{2}  
Adam Gleave\textsuperscript{3}  
Anssi Kanervisto\textsuperscript{4}  
Maximilian Ernestus\textsuperscript{5}  
Noah Dormann\textsuperscript{1}  

\textsuperscript{1} Robotics and Mechatronics Center (RMC), German Aerospace Center (DLR), Weßling, Germany  
\textsuperscript{2} Interactive Robotics Laboratory, University Paris-Saclay, CEA, Palaiseau, France  
\textsuperscript{3} Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA  
\textsuperscript{4} School of Computing, University of Eastern Finland, Joensuu, Finland  
\textsuperscript{5} Kiteswarms GmbH, Freiburg, Germany  

Editor: Andreas Mueller

Abstract

Stable-Baselines3 provides open-source implementations of deep reinforcement learning (RL) algorithms in Python. The implementations have been benchmarked against reference codebases, and automated unit tests cover 95\% of the code. The algorithms follow a consistent interface and are accompanied by extensive documentation, making it simple to train and compare different RL algorithms. Our documentation, examples, and source-code are available at \url{https://github.com/DLR-RM/stable-baselines3}.  

Keywords: Reinforcement Learning, Baselines, Software, Open-Source, Python, PyTorch

1. Introduction

Deep reinforcement learning (RL) research has grown rapidly in recent years, yet results are often difficult to reproduce (Henderson et al., 2018). A major challenge is that small implementation details can have a substantial effect on performance – often greater than the difference between algorithms (Engstrom et al., 2020). It is particularly important that implementations used as experimental baselines are reliable; otherwise, novel algorithms compared to weak baselines lead to inflated estimates of performance improvements.

To address this challenge, we propose Stable-Baselines3 (SB3), an open-source framework implementing seven commonly used model-free deep RL algorithms (see Section 2). We take great care to adhere to software engineering best practices to achieve high-quality implementations that match prior results. Each algorithm has been benchmarked on common environments (Raffin and Stulp, 2020) and compared to prior implementations. Our test suite covers 95\% of the code and, together with our active user base\textsuperscript{1} scrutinizing changes, ensures that any implementation errors are minimized.

\textsuperscript{1} At the time of writing, SB3 had 800+ stars on GitHub, 100+ closed issues and 80+ merged pull requests

\(^{c\textcopyright}2021\) Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus and Noah Dormann.  
License: CC-BY 4.0, see \url{https://creativecommons.org/licenses/by/4.0/}. Attribution requirements are provided at \url{http://jmlr.org/papers/v22/20-1364.html}.  

import gym
from stable_baselines3 import SAC
# Train an agent using Soft Actor-Critic on Pendulum-v0
env = gym.make("Pendulum-v0")
model = SAC("MlpPolicy", env).learn(total_timesteps=20000)
# Save the model
model.save("sac_pendulum")
# Load the trained model
model = SAC.load("sac_pendulum")
# Start a new episode
obs = env.reset()
# What action to take in state `obs`?
action, _ = model.predict(obs, deterministic=True)

Figure 1: Using Stable-Baselines3 to train, save, load, and infer an action from a policy.

Stable-Baselines3 builds on the experience gained from maintaining our previous implementation, Stable-Baselines2 (SB2; Hill et al., 2018), that was forked from OpenAI Baselines (Dhariwal et al., 2017) and uses TensorFlow (Abadi et al., 2016). SB3 is a complete rewrite of the codebase implemented in PyTorch (Paszke et al., 2019), the framework preferred by a majority of our users in a survey (Raffin, 2020a). SB3 maintains a similar API, allowing a seamless upgrade pathway from SB2.

Our main goal is to provide a user-friendly and reliable RL library. To keep SB3 simple to use and maintain, we focus on model-free, single-agent RL algorithms, and rely on external projects to extend the scope to imitation (Wang et al., 2020) and offline learning (Seno, 2020). We prioritize maintaining stable implementations over adding new features or algorithms, and avoid making breaking changes. We provide a consistent, clean and fully documented API, inspired by the scikit-learn API (Pedregosa et al., 2011). Our code is easily modifiable by users as we favour readability and simplicity over modularity, although we make use of object-oriented programming to reduce code duplication.

2. Features

Simple API. Figure 1 shows that training agents in Stable-Baselines3 takes just a few lines of code, after which the agent can be queried for actions. This allows researchers to easily use the baseline algorithms and components in their experiments (e.g. Klink et al. (2020); Nair et al. (2019); Gleave et al. (2020)), as well as apply RL to novel tasks and environments, like continual learning when attacking WiFi networks (Margaritelli, 2020) or dampening bridge vibrations (Berkowitz, 2019).

Documentation. SB3 comes with extensive documentation of the code API. We also include a user guide, covering both basic and more advanced usage with a collection of concrete examples. Moreover, we have developed a Colab notebook based RL tutorial, enabling users to demo the library directly in the browser. Additionally, we include common

---

2. SB2 has 650+ closed issues, 220+ merged pull requests and 200+ citations on Google Scholar.
Stable-Baselines3: Reliable Reinforcement Learning Implementations

tips for running RL experiments and a developer guide. We also pay close attention to
questions and uncertainties from SB3 users, updating the documentation to address these.

**High-Quality Implementations.** Algorithms are verified against published results
by comparing the agent learning curves\(^6\). Moreover, all functions are typed (parameter and
return types) and documented with a consistent style, and most functions are covered by
unit tests. Continuous integration checks that all changes pass unit tests and type check,
as well as validating the code style and documentation.

**Comprehensive.** Stable-Baselines3 contains the following state-of-the-art on- and
off-policy algorithms, commonly used as experimental baselines: A2C (Mnih et al., 2016),
PPO (Schulman et al., 2017), DDPG (Lillicrap et al., 2016), SAC (Haarnoja et al., 2018),
TD3 (Fujimoto et al., 2018), HER (Andrychowicz et al., 2017) and DQN (Mnih et al., 2015).

Moreover, SB3 provides various algorithm-independent features. We support logging
to CSV files and TensorBoard. Users can log custom metrics and modify training via
user-provided callbacks. To speed up training, we support parallel (or “vectorized”) envi-
ronments. To simplify training, we implement common environment wrappers, like prepro-
cessing Atari observations to match the original DQN experiments (Mnih et al., 2015).

**Experimental Framework.** RL Baselines Zoo (Raffin, 2018, 2020b) provides scripts
to train and evaluate agents, tune hyperparameters, record videos, store experiment setup
and visualize results. We also include a collection of pre-trained reinforcement learning
agents together with tuned hyperparameters for simple control tasks, PyBullet environ-
ments (Coumans and Bai, 2016–2019) and Atari games, optimized using Optuna (Akiba
et al., 2019). We follow best practices for training and evaluation (Henderson et al., 2018),
such as evaluating in a separate environment, using deterministic evaluation where required
(SAC) and storing all hyperparameters necessary to replicate the experiment.

**Stable-Baselines3 Contrib.** We implement experimental features in a separate con-
trib repository (Raffin et al., 2020). This allows SB3 to maintain a stable and compact
core, while still providing the latest features, like Truncated Quantile Critics (Kuznetsov
et al., 2020). Implementations in contrib need not be tightly integrated with the main SB3
codebase, but we maintain the same stringent review requirements to ensure users can trust
the contrib implementations. Implementations from contrib that have stood the test of time
may be integrated into the main repository.

3. **Comparison to Related Software**

Most libraries are targeted at experienced RL researchers, requiring expert knowledge to
use (Weng et al., 2020; Hoffman et al., 2020; Fujita et al., 2021; Castro et al., 2018; Guadarr-
rama et al., 2018; Gauci et al., 2018; Stooke and Abbeel, 2019; Kolesnikov, 2018). Only
a few RL libraries offer more than a brief API documentation (garage contributors, 2019;
Liang et al., 2018; Kuhnle et al., 2017; Guadarrama et al., 2018), and some are notoriously
hard to understand.\(^7\) By contrast, Stable-Baselines3 is designed to be easy to use and
comes with extensive documentation and tutorials.

---

6. For example, issue \#48 or issue \#49.

7. OpenAI Baselines (Dhariwal et al., 2017), see [https://www.reddit.com/r/MachineLearning/comments/95ft1j/](https://www.reddit.com/r/MachineLearning/comments/95ft1j/). This was a major starting point for Stable-Baselines2 (Hill et al., 2018)
The previous version of Stable-Baselines3, Stable-Baselines2, was created as a fork of OpenAI Baselines (Dhariwal et al., 2017) but the two codebases quickly diverged (see PR #481). SB3 is a complete rewrite of Stable-Baselines2 in PyTorch that keeps the major improvements and new algorithms from SB2 while going even further into improving code quality (e.g. cleaner codebase, better test coverage, type hints). More precisely, compared to Baselines, SB3 is fully documented, commented, tested, has 4 additional algorithms (SAC, TD3, QR-DQN, TQC) and many additional features (e.g. dictionary observation support, callbacks, evaluation with multiple environments, environment checker). The only legacy features of OpenAI Baselines are the code structure (one folder per algorithm), the use of code-level optimizations and the environment tools which are greatly improved (additional features, bug fixes, comments, documentation and more testing).

Many libraries have a modular design (Caspi et al., 2017; Keng and Graesser, 2017; Hoffman et al., 2020; garage contributors, 2019). This allows them to quickly combine advances from different papers, but forces new users to understand the full code structure before being able to tweak the library. On the other extreme, educational implementations like Spinning Up (Achiam, 2018) are self-contained but hard to maintain due to code duplication. SB3 strives to strike a balance: factoring out widely used components like replay buffers, but minimizing the amount of code that needs to be understood to modify an algorithm.

As an exhaustive comparison to all RL libraries is not possible, in Table 1 we compare SB3 to a subset of other active or popular libraries, with a focus on quality of implementation and openness to new users.

RLlib (Liang et al., 2018) scores highly in the table, but is targeted at a different use-case from SB3. Whereas SB3 focuses on simplicity and reliability, RLlib (Liang et al., 2018) prioritizes scalability and support for distributed training. Additionally, RLlib includes both a PyTorch and TensorFlow backend, and includes support for multi-agent training. This versatility comes at a cost of a larger and more complex codebase.

Overall, we find SB3 compares favourably to other libraries in terms of documentation, testing and activity.

<table>
<thead>
<tr>
<th>SB3</th>
<th>OAI Baselines</th>
<th>PFRL</th>
<th>RLlib</th>
<th>Tianshou</th>
<th>Acme</th>
<th>Tensorforce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backend</td>
<td>PyTorch</td>
<td>TF</td>
<td>PyTorch/TF</td>
<td>PyTorch</td>
<td>Jax/TF</td>
<td>TF</td>
</tr>
<tr>
<td>User Guide / Tutorials</td>
<td>✓/✓</td>
<td>✗/+</td>
<td>✓/✓</td>
<td>✓/+</td>
<td>✓/+</td>
<td>✓/+</td>
</tr>
<tr>
<td>API Documentation</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Benchmark</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Pretrained models</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Test Coverage</td>
<td>95%</td>
<td>49%</td>
<td>?</td>
<td>?</td>
<td>94%</td>
<td>74%</td>
</tr>
<tr>
<td>Type Checking</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Issue / PR Template</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Last Commit (age)</td>
<td>&lt; 1 week</td>
<td>&gt; 6 months</td>
<td>&lt; 1 month</td>
<td>&lt; 1 week</td>
<td>&lt; 1 month</td>
<td>&lt; 1 week</td>
</tr>
<tr>
<td>Approved PRs (6 mo.)</td>
<td>75</td>
<td>0</td>
<td>13</td>
<td>222</td>
<td>85</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 1: Comparison of SB3 to a representative subset of active or popular RL libraries. Key: ✗ means that the feature is only partially present; OAI: OpenAI; TF: TensorFlow; PR: Pull Request.

8. QR-DQN and TQC are in the contrib repo.
9. As an example, one can compare “VecNormalize” in OAI Baselines vs SB3.
Acknowledgments

The work described in this paper was partially funded by the project “Reduced Complexity Models” from the “Helmholtz-Gemeinschaft Deutscher Forschungszentren” and by the EU H2020 project “VERtical Innovation in the Domain of Robotics Enabled by Artificial intelligence Methods”.

References


