SLM Lab:

Modular Deep RL framework in PyTorch

Wah Loon Keng (kengzwl@gmail.com) Laura Graesser (lhgraesser@gmail.com)

Abstract

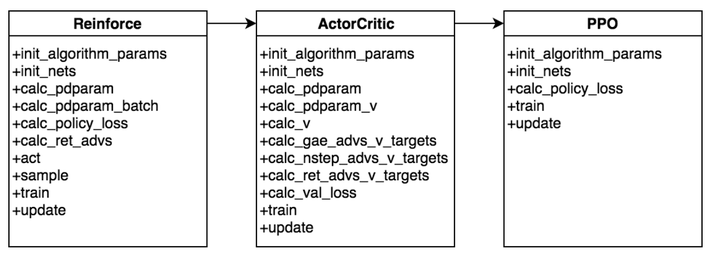
We introduce SLM Lab, a software framework for reproducible reinforcement learning (RL) research. SLM Lab implements a number of popular RL algorithms, provides synchronous and asynchronous parallel experiment execution, hyperparameter search, and result analysis. RL algorithms in SLM Lab are implemented in a modular way such that differences in algorithm performance can be confidently ascribed to differences between algorithms, not between implementations.

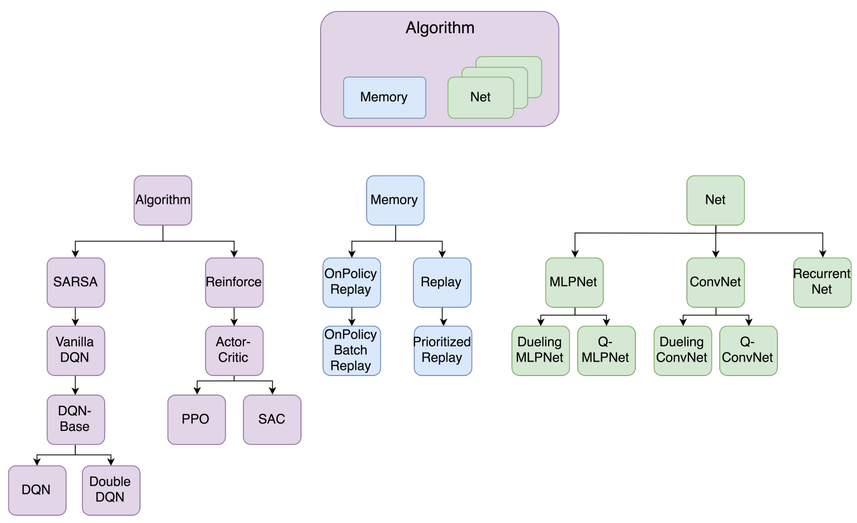
github.com/kengz/SLM-Lab

Modular Design

Modular code is critical for deep RL research because many algorithms are extensions of other algorithms. If two RL algorithms differ in only a small way, so should their implementations — by reusing modular components. Modular design also make it as simple as possible for a researcher to implement and reliably evaluate new RL algorithms, and for the student of RL to learn from lighter, well-organized code. The implemented algorithms in SLM Lab are:

- SARSA
- DQN
- Double DQN, Dueling-DQN, PER
- REINFORCE
- A2C with GAE and n-step
- PPO (Proximal Policy Optimization)
- SAC (Soft Actor-Critic)

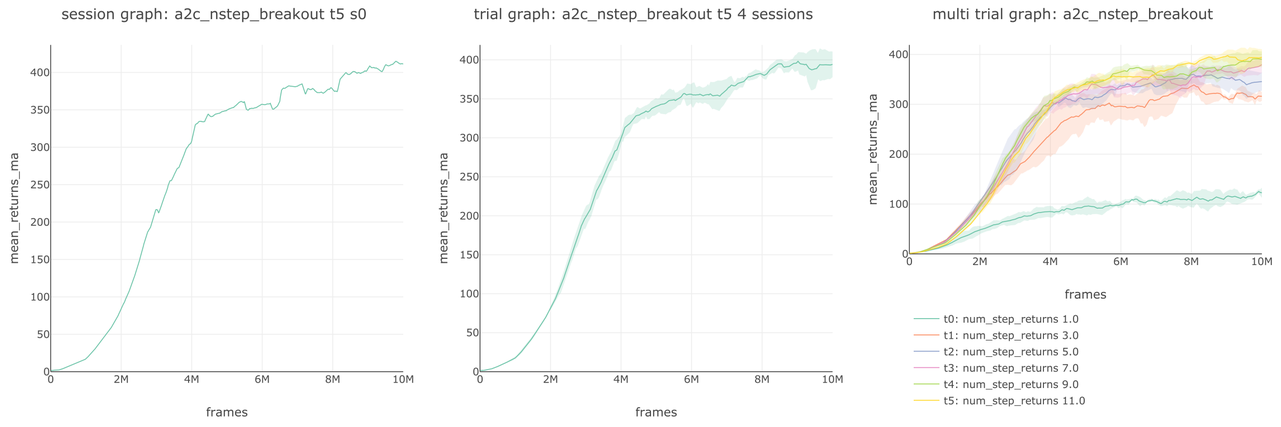




Experiment Organization

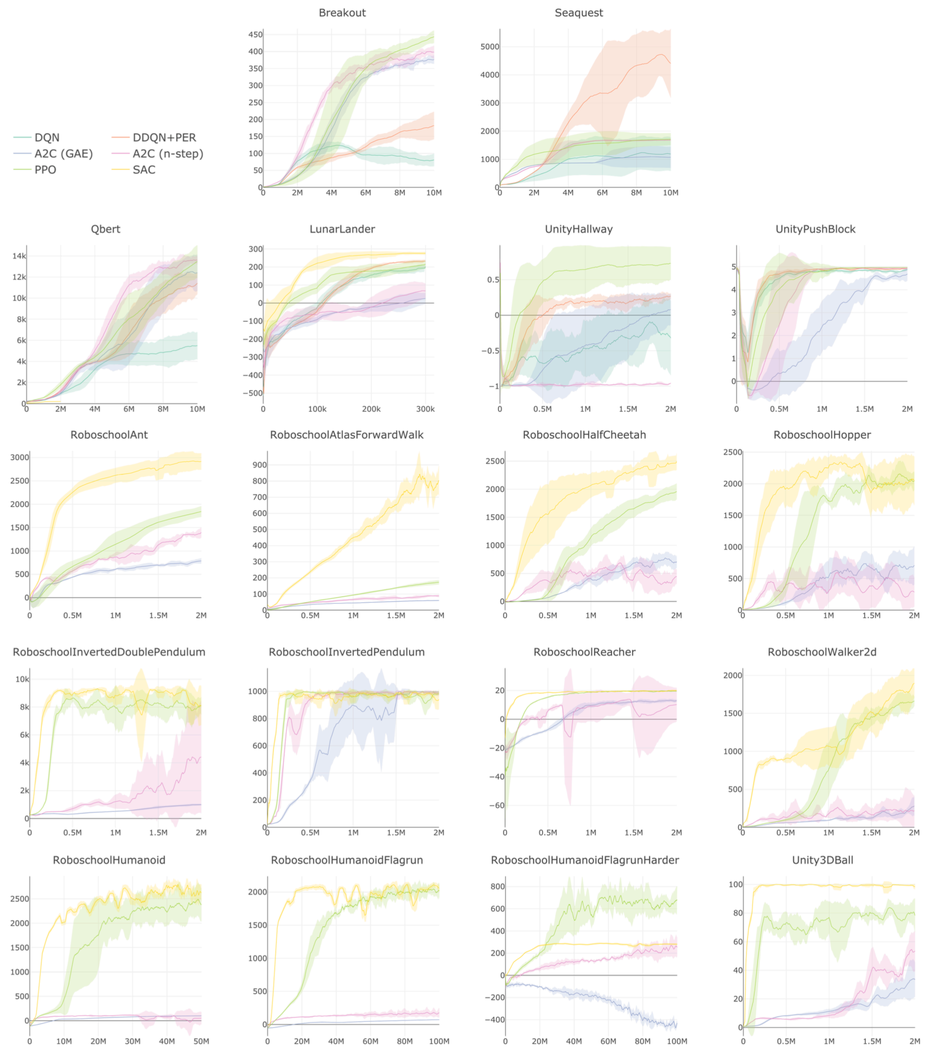
RL algorithms vary greatly in their performance across different environments, hyperparameter settings, and even within a single environment due to inherent randomness. SLM Lab is designed to easily allow users to study all these types of variability with the following experiment organization:

1. **Session:** a single training run of a one agent on one environment with one set of hyperparameters, all with a fixed random seed.
2. **Trial:** consists of multiple Sessions, with the Sessions varying only in the random seed.
3. **Experiment:** generates different sets of hyperparameters and runs a Trial for each one. It can be thought of as a study, e.g. “What values of “n” of A2C n-step returns gives the best performance, if the other variables are held constant?”



Benchmark Results

Here are some benchmark results on discrete and continuous environments. Reported scores are the average over the last 100 checkpoints, averaged over 4 Sessions. A larger set of results, including the full Atari suite, is available on Github.



** trained using asynchronous SAC to speed up training time.*

Environment	Algorithm				
	DQN	DDQN+PER	A2C (GAE)	A2C (n-step)	PPO
Breakout	80.88	182	377	398	**443**
Pong	18.48	20.5	19.31	19.56	**20.58**
Seaquest	1185	**4405**	1070	1684	1715
Qbert	5494	11426	12405	**13590**	13460
LunarLander	192	233	25.21	68.23	214
UnityHallway	-0.32	0.27	0.08	-0.96	**0.73**
UnityPushBlock	4.88	4.93	4.68	4.93	**4.97**

Environment	Algorithm			
	A2C (GAE)	A2C (n-step)	PPO	SAC
RoboschoolAnt	787	1396	1843	**2915**
RoboschoolAtlasForwardWalk	59.87	88.04	172	**800**
RoboschoolHalfCheetah	712	439	1960	**2497**
RoboschoolHopper	710	285	2042	**2045**
RoboschoolInvertedDoublePendulum	996	4410	8076	**8085**
RoboschoolInvertedPendulum	**995**	978	986	941
RoboschoolReacher	12.9	10.16	19.51	**19.99**
RoboschoolWalker2d	280	220	1660	**1894**
RoboschoolHumanoid	99.31	54.58	2388	**2621***
RoboschoolHumanoidFlagrun	73.57	178	2014	**2056***
RoboschoolHumanoidFlagrunHarder	-429	253	**680**	280*
Unity3DBall	33.48	53.46	78.24	**98.44**
Unity3DBallHard	62.92	71.92	91.41	**97.06**