The Journal of Open Source Software

RL: Generic reinforcement learning codebase in TensorFlow

Bryan M. Li[1], Alexander Cowen-Rivers[1], Piotr Kozakowski[1], David Tao[1], Siddhartha Rao Kamalakara[1], Nitarshan Rajkumar[1], Hariharan Sezhiyan[1], Sicong Huang[1], and Aidan N. Gomez[1]

1 FOR.ai

DOI: 10.21105/joss.01524

Software

- Review
- Repository
- Archive

Submitted: 13 June 2019
Published: 04 October 2019

Abstract

Vast reinforcement learning (RL) research groups, such as DeepMind and OpenAI, have their internal (private) reinforcement learning codebases, which enable quick prototyping and comparing of ideas to many state-of-the-art (SOTA) methods. We argue the five fundamental properties of a sophisticated research codebase are: modularity, reproducibility, many RL algorithms pre-implemented, speed and ease of running on different hardware/ integration with visualization packages. Currently, there does not exist any RL codebase, to the author's knowledge, which contains all the five properties, particularly with TensorBoard logging and abstracting away cloud hardware such as TPU's from the user. The codebase aims to help distil the best research practices into the community as well as ease the entry access and accelerate the pace of the field. More detailed documentation can be found here.

Related Work

There are currently various implementations available for reinforcement learning codebase like OpenAI baselines (Dhariwal et al., 2017), Stable baselines (Hill, 2019), Tensorforce (Schaarschmidt, Kuhnle, & Fricke, 2017), Ray rllib (Liang et al., 2017), Intel Coach (Caspi, Leibovich, & Novik, 2017), Keras-RL (Plappert, 2019), Dopamine baselines (Castro, Moitra, Gelada, Kumar, & Bellemare, 2018) and TF-Agents (Sergio Guadarrama, 2018). Ray rllib (Liang et al., 2017) is amongst the strongest of existing RL frameworks, supporting; distributed operations, TensorFlow (Abadi et al., 2016), PyTorch (Paszke et al., 2017) and multi-agent reinforcement learning (MARL). Unlike Ray rllib, we choose to focus on Tensorflow support, allowing us to integrate specific framework visualisation and experiment tracking into our codebase. On top of this, we are developing a Kuberenetes script for MacOS and Linux users to connect to any cloud computing platform, such as Google TPU's, Amazon AWS etc. Most other frameworks are plagued with problems like usability issues (difficult to get started and increment over), very little modularity in code (no/ little hierarchy and code reuse), no asynchronous training support, weak support for TensorBoard logging and so on. All these problems are solved by our project, which is a generic codebase built for reinforcement learning (RL) research in Tensorflow (Schaarschmidt et al., 2017), with favoured RL agents pre-implemented as well as integration with OpenAI Gym (Brockman et al., 2016) environment focusing on quick prototyping and visualisation.

Deep Reinforcement Learning Reinforcement learning refers to a paradigm in artificial intelligence where an agent performs a sequence of actions in an environment to maximise rewards (Sutton & Barto, 1998). It is in many ways more general and challenging than supervised learning since it requires no labels to train on; instead, the agent interacts continuously with the environment, gathering more and more data and guiding its learning process.

Introduction: for-ai/rl

Further to the core ideas mentioned in the beginning, a good research codebase should enable good development practices such as continually checkpointing the model's parameters as well as instantly restoring them to the latest checkpoint when available. Moreover, it should be composed of simple, interchangeable building blocks, making it easy to understand and to prototype new research ideas.

We will first introduce the framework for this project, and then we will detail significant components. Lastly, we will discuss how to get started with training an agent under this framework.

This codebase allows training RL agents by a training script as simple as the below for loop.

```
for epoch in range(epochs):
    state = env.reset()
    for step in range(max_episode_steps):
        last_state = state
        action = agent.act(state)
        state, reward, done = env.step(action)
        agent.observe(last_state, action, reward, state)
        agent.update()
```

To accomplish this, we chose to modularise the codebase in the hierarchy shown below.

```
rl_codebase
|- train.py
|---> memory
|   |- registry.py
|---> hparams
|   |- registry.py
|---> envs
|   |- registry.py
|---> models
|   |- registry.py
|---> agents
|   |- registry.py
|   |---> algos
|   |   |- registry.py
|   |   |---> act_select
|   |   |   |- registry.py
|   |   |---> grad_comp
|   |   |   |- registry.py
```

Our modularisation enables simple and easy-to-read implementation of each component, such as the Agent, Algo and Environment class, as shown below.

```
class Agent:
    self.model: Model
    self.algo: Algo

    def observe(last_state, action, reward, new_state)
    def act(state) -> action
    def update()
```

```
class Algo(Agent):
    def select_action(distribution) -> action
    def compute_gradients(trajectory, parameters) -> gradients

class Environment:
    def reset() -> state
    def step(action) -> state, reward, done
```

The codebase includes agents like Deep Q Network (Mnih et al., 2013), Noisy DQN (Plappert et al., 2017), Vanilla Policy Gradient (**???**), Deep Deterministic Policy Gradient (Silver et al., 2014) and Proximal Policy Optimization (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). The project also includes simple random sampling and proportional prioritized experience replay approaches, support for Discrete and Box environments, option to render environment replay and record the replay in a video. The project also gives the possibility to conduct model-free asynchronous training, setting hyperparameters for your algorithm of choice, modularized action and gradient update functions and option to show your training logs in a TensorBoard summary.

In order to run an experiment, run:

`python train.py --sys ... --hparams ... --output_dir ....`

Ideally, "train.py" should never need to be modified for any of the typical single agent environments. It covers the logging of reward, checkpointing, loading, rendering environment/ dealing with crashes and saving the experiments hyperparameters, which takes a significant workload off the average reinforcement learning researcher.

```
"--sys"(str) defines the system chosen to run experiment with;
  e.g. "local" for running on the local machine.
"--env"(str) specifies the environment.
"--hparam_override"(str) overrides hyperparameters.
"--train_steps"(int) sets training length.
"--test_episodes"(int) tests episodes.
"--eval_episodes"(int) sets Validation episodes.
"--training"(bool) freeze model weights is set to False.
"--copies"(int) set the number of times to perform multiple
  versions of training/testing.
"--render"(bool) turns rendering on/ off.
"--record_video"(bool) records the video with, which outputs a .mp4
  of each recorded episode.
"--num_workers"(int) seamlessly brings our synchronous agent
  into an asynchronous agent.
```

Full Example

Before you run a full examples, it would be to your benefit to install the following:

- Nvidia CUDA on machines with GPUs to enable faster training. Installation instructions here
- Tensorboard for training visualization. Install by running `pip install tensorboard`

This tuturial will make use of a Conda environment as the preferred package manager. Installation instructions can be found here.

After installing Conda, create and activate an environment, and install all dependencies within that environment:

```
conda create -n rl python=3.6
conda activate rl
sh setup.sh
```

To run locally, we will train DQN on the `Carpole-v1` Gym environment:

```
# start training
python train.py --sys local --hparams dqn_cartpole --output_dir /tmp/rl-testing
# run tensorboard
tensorboard --logdir /tmp/rl-testing
# test agent
python train.py --sys local --hparams dqn_cartpole --output_dir /tmp/rl-testing
  --training False --render True
```

Conclusion

We have outlined the benefits of using a highly modularised reinforcement learning codebase. The next stages of development for the RL codebase are implementing more SOTA model-free RL techniques (GAE, Rainbow, SAC, IMPALA), introducing model-based approaches, such as World Models (Ha & Schmidhuber, 2018), integrating into an open-sourced experiment managing tool and expanding the codebases compatibility with a broader range of environments, such as Habitat (Savva et al., 2019). We would also like to see automatic hyperparameter optimization techniques to be integrated, such as Bayesian Optimization method which was crucial to the success of some of DeepMinds most considerable reinforcement learning feats (Y. Chen et al., 2018).

Acknowledgements

We would like to thank all other members of For.ai, for useful discussions and feedback.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th usenix symposium on operating systems design and implementation osdi 16)* (pp. 265–283).

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Caspi, I., Leibovich, G., & Novik, G. (2017). Reinforcement learning coach.(Dec. 2017), *1134899*. doi:10.5281/zenodo.1134899

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., & Bellemare, M. G. (2018). Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*.

Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., & Freitas, N. de. (2018). Bayesian optimization in alphago. *arXiv preprint arXiv:1812.06855*.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., et al. (2017). Openai baselines. Retrieved from https://github.com/openai/baselines

Ha, D., & Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In *Advances in neural information processing systems* (pp. 2450–2462).

Hill. (2019, June). Hill-a/stable-baselines. Retrieved from https://github.com/hill-a/stable-baselines

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Gonzalez, J., Goldberg, K., et al. (2017). Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., et al. (2017). Automatic differentiation in pytorch. In *NIPS autodiff workshop*.

Plappert, M. (2019, March). Keras-rl/keras-rl. *GitHub*. Retrieved from https://github.com/keras-rl/keras-rl

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., et al. (2017). Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., et al. (2019). Habitat: A platform for embodied ai research. *arXiv preprint arXiv:1904.01201*.

Schaarschmidt, M., Kuhnle, A., & Fricke, K. (2017). Tensorforce: A tensorflow library for applied reinforcement learning. Retrieved from https://github.com/tensorflow/tensorflow

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sergio Guadarrama, O. R., Anoop Korattikara. (2018). TF-agents: A library for reinforcement learning in tensorflow. Retrieved from "https://github.com/tensorflow/agents"

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. In E. P. Xing & T. Jebara (Eds.), *Proceedings of the 31st International Conference on Machine Learning*, Proceedings of machine learning research (Vol. 32, pp. 387–395). Bejing, China: PMLR. Retrieved from http://proceedings.mlr.press/v32/silver14.html

Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning. Vol. 135. Cambridge: MIT press.