rl-codebase

Release 0.0.1

FOR.ai

Aug 06, 2019

CONTENTS

Contents:

CHAPTER

ONE

OVERVIEW

1.1 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 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.

1.2 Related Work

There are currently various implementations available for reinforcement learning codebase like OpenAI baselines [DHK+17], Stable baselines [Hil19], Tensorforce [SKF17], Ray rllib [LLN+17], Intel Coach [CLN17], Keras-RL [KR19], Dopamine baselines [CMG+18] and TF-Agents [GKR+]. Ray rllib [LLN+17] is amongst the strongest of existing RL frameworks, supporting; distributed operations, TensorFlow [ABC+16], PyTorch [PGCC17] 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 [SKF17], with favoured RL agents pre-implemented as well as integration with OpenAI Gym [BCP+16] 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 [SB98]. 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.

1.3 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 [MKS+13], Noisy DQN [PHD+17], Vanilla Policy Gradient [SMSM00], Deep Deterministic Policy Gradient [SLH+14] and Proximal Policy Optimization [SWD+17]. 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.

Below we summerize the key arguments

```
"--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.
```

1.4 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 [HS18], integrating into an open-sourced experiment managing tool and expanding the codebases compatibility with a broader range of environments, such as Habitat [SKM+19]. 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 [CHW+18].

1.5 Acknowledgements

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

1.6 References

CHAPTER

TWO

INSTALLATION

Install all dependencies by running:

```
$ ./setup.sh
```

This installation script currently supports Linux and macOS. For macOS users, this installation script supports both Homebrew and Macports. In the event the script fails, we recommend installing all dependencies within a Conda environment.

CHAPTER

THREE

MODULES

This section covers the main components of the RL codebase.

3.1 Agents

All agents are encapsulated into a single class, `Agent()`. An agent object is instantiated with the following member variables:

- TensorFlow session
- Hyperparameters
- Memory
- Action function
- Gradient function

3.2 Algos

This codebase currently supports the following RL algorithms: `DDPG`, `DQN`, `PPO`, and `Vanilla Policy Gradient`.

3.3 Environments

The codebase has a single environment class, and two environment subclasses:

- **`class Environment`**
  - `class CoinRun(Environment)` as described here
  - `class GymEnv(Environment)` as descibed here

The `Environment` class, located at `rl/rl/envs/env.py`, acts as a template contains all the member variables and functions of a standard RL environment. The specific member variables and functions are implemented within the two subclasses.

3.4 Hyperparameters

The class to represent hyperparameters, `HParams`, is located at `rl/rl/hparams/utils.py`. The default set of hyperparameters located at `rl/rl/hparams/default.py`.

The list of hyperparameters is located at `rl/rl/hparams`, segmented into the different RL algorithms each set of hyperparameters belongs to. Each RL algorithms calls the default set of hyperparameters, and depending on different environments, modifies the default set.

For example, the function `ddpg()` returns a default set of `HParams` and adds a number of fields, while the function `ddpg_cartpole()` further modifies some of the hyperparameter fields to fit with the `Cartpole-v1` Gym environment

3.5 Memory

The base class for memory is located at `rl/rl/memory/memory.py`. This codebase currently supports three memory types: simple experience reply, sum tree, and prioritized experience replay.

3.6 Models

The list of models can be found at `rl/rl/models/models`. All models are subclassed from Keras models. Each model is instantiated with a network structure and a `call()` function to call the forward pass of the network. All layers are taken from the TensorFlow Layers class.

CHAPTER

FOUR

TUTORIAL

Before you run a full example, 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-codebase python=3.6
$ conda activate rl-codebase
$ pip install -r requirements.txt
```

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
```

CHAPTER FIVE

USAGE

To start training:

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

Run tensorboard to visualize training:

```
$ tensorboard --logdir ...
```

Test agent:

```
$ python train.py --sys ... --hparams ... --output_dir ... --training False --render␣
↪True
```

The sys command can be one of two options: `local` or `tpu` for GCP enabled tpu training. A list of environments and hyperparameters can be found under `rl/rl/hparams`. A full training and evaluation example can be found in the tutorial section.

Below we summerize the key arguments

```
“--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.
```

BIBLIOGRAPHY

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, and others. Tensorflow: a system for large-scale machine learning. In *12th $\$USENIX$\$ Symposium on Operating Systems Design and Implementation ($\$OSDI$\$ 16)*, 265–283. 2016.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[CLN17] Itai Caspi, Gal Leibovich, and Gal Novik. Reinforcement learning coach.(dec. 2017). *URL https://doi.org/10.5281/zenodo*, 2017.

[CMG+18] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: a research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

[CHW+18] Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver, and Nando de Freitas. Bayesian optimization in alphago. *arXiv preprint arXiv:1812.06855*, 2018.

[DHK+17] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. *GitHub, GitHub repository*, 2017.

[GKR+] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Chris Harris, Vincent Vanhoucke, and others. Tf-agents: a library for reinforcement learning in tensorflow.

[HS18] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, 2450–2462. 2018.

[Hil19] Hill. Hill-a/stable-baselines. Jun 2019. URL: https://github.com/hill-a/stable-baselines.

[KR19] Matthias Plappert Keras-Rl. Keras-rl/keras-rl. Mar 2019. URL: https://github.com/keras-rl/keras-rl.

[LLN+17] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Ken Goldberg, and Ion Stoica. Ray rllib: a composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*, 2017.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[PGCC17] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. Pytorch. *Computer software. Vers. 0.3*, 2017.

[PHD+17] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[SKM+19] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, and others. Habitat: a platform for embodied ai research. *arXiv preprint arXiv:1904.01201*, 2019.

[SKF17] Michael Schaarschmidt, Alexander Kuhnle, and Kai Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. *Web page*, 2017.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[SLH+14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*. 2014.

[SB98] Richard S Sutton and Andrew G Barto. Introduction to reinforcement learning. vol. 135. 1998.

[SMSM00] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063. 2000.