UDACITY

‹ Return to "Deep Reinforcement Learning Nanodegree" in the classroom

Collaboration and Competition

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations 👏🎉 You passed this Project, that too on your first attempt. I hope you had great experience doing this project and will continue this Nanodegree and take more Nanodegrees in the future. I have provided a few tips for further improving upon this project.

If you wish to study the subject further, here are few of my recommendations:

1. A (Long) Peek into Reinforcement Learning briefly go over the field of Reinforcement Learning (RL), from fundamental concepts to classic algorithms [WARNING] This is a long read.
2. Policy Gradient Algorithms Looks deep into policy gradient, why it works, and many new policy gradient algorithms proposed in recent years: vanilla policy gradient, actor-critic, off-policy actor-critic, A3C, A2C, DPG, DDPG, D4PG, MADDPG, TRPO, PPO, ACER, ACTKR, and Soft AC.

Experience Sampling

1. Prioritised Experience Replay paper
2. Hindsight Experience Replay paper OpenAI Blog covering HER

Hyperparameter Search

There're mainly 2-3 most popular methods of doing hyperparameter search:

- Random search: It's still sometime considered the king in hyperparameter search despite the performance of other methods. Knowing the minimum and maximum range for some parameter due to prior knowledge is always helpful in narrowing the range.

- Hyperband: It's a bandit-based approach and I'd recommend reading this co-author's blog
- Bayesian Optimisation Techniques: These are mostly covered in the above blog but are also available through frameworks (some also have HyperBand) like Hyperopt and Ray-Tune.
- Also, I'd recommend going through the the wiki page for more types of optimisation.

Training Code

The repository includes functional, well-documented, and organized code for training the agent.

You have included a well-organized zip file with README.md, a jupyter notebook, and the code files. I was able to run the code to instantiate the agents and train them.

The code is written in PyTorch and Python 3.

Your code is written in Python and Pytorch.

The submission includes the saved model weights of the successful agent.

You've included the saved model weights in the submission. The reason we ask for this & why this is important is simple: "memory is much cheaper than compute". The training time used is significantly more than a few MBs of storage. Thus, this helps in verifying as well as building uupon your previous experiments.

README

The GitHub submission includes a `README.md` file in the root of the repository.

You've included a README.md file in your submission. This is a crucial as:

- The Readme is often the first look someone has at your project.
- It helps me communicate project goals to others
- You can also have a basic documentation for things other than getting started (or install instructions) like Contribution guidelines, Contributor list, contact and so on.

The README describes the the project environment details (i.e., the state and action spaces, and when the environment is considered solved).

You correctly described the environment details. Also, it includes the description of the action and state spaces, how the reward is decided, and when the environment is considered solved.

The README has instructions for installing dependencies or downloading needed files.

Instructions are included for setting up.

The README describes how to run the code in the repository, to train the agent. For additional resources on creating READMEs or using Markdown, see here and here.

You provided detailed instructions on how to run the agent.

Report

The submission includes a file in the root of the GitHub repository (one of `Report.md`, `Report.ipynb`, or `Report.pdf`) that provides a description of the implementation.

Good Job! You've clearly described the learning algorithm, the chosen hyperparameters along with the model architectures for neural networks in the report.

The report clearly describes the learning algorithm, along with the chosen hyperparameters. It also describes the model architectures for any neural networks.

Good Job! You've clearly described the learning algorithm.

A plot of rewards per episode is included to illustrate that the agents get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents).

The submission reports the number of episodes needed to solve the environment.

You've included the plot of rewards in `Ipynb` notebook.
Excellent work! Your agent was able to solve the environment in a decent number of episodes!

The submission has concrete future ideas for improving the agent's performance.

Nice work! These are very good ideas. Perhaps as an optional challenge, you could try PER and see if (by how much?) it improves the performance and maybe report them in a blog post or your repo.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review