UDACITY

‹ Return to Classroom

Continuous Control

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Dear Udacian,

Great job getting acquainted with the Deep Deterministic Policy Gradients algorithm and implementing it to successfully solve the Reacher environment. The implementation is pretty good and the agent achieves an average score of +36 from 37 to 136 episodes. The architectures used for the actor and critic networks are good in size with two hidden layers each. Good work using `relu` and `leaky_relu` activations in the actor and critic networks, respectively. The report is extremely informative and covers all the important aspects of the implementation. 😄

I would suggest you to go through Deep Reinforcement Learning for Self Driving Car by MIT. You'd get to know more about reinforcement learning algorithms in broader and real-world perspective and, more importantly, how to apply these techniques to real-world problems.

All the best for future endeavors. ✨

Training Code

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

Awesome

- Good work implementing DDPG algorithm to solve robotic-arms Reacher environment.
- Implementation of the Actor and Critic networks is correct.
- Good work using the target networks for Actor and Critic networks. The original DDPG paper suggests it as well.
- Good work using soft updates for the target network.
- Good choice to use tau to perform soft update.
- Correct usage of replay memory to store and recall experience tuples.
- The implementation is easy to debug and easily extensible, good work keeping it highly modular.

The code is written in PyTorch and Python 3.

Awesome

The code is written in PyTorch and Python 3.

Lately, PyTorch and TensorFlow happen to be most extensively used frameworks in deep learning. It would be good to get some insight by comparing them, please see the following resources:

- Sebastian Thrun on TensorFlow
- PyTorch vs TensorFlow—spotting the difference
- Tensorflow or PyTorch : The Force is Strong with which One?

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

Awesome

- Saved model weights of the successful agent have been submitted.
- `checkpoint_actor.pth` and `checkpoint_critic.pth` files are present in the submission.

README

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

Awesome

- Great work documenting the project details and submitting the README file.

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

Awesome

- Great work providing the details of the project environment in the `Introduction` section of the README.
- The section describes the project environment by specifying the state space, action space, and the desired results.

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

Awesome

- Great work providing the all the necessary instructions in the `Getting Started` section to download the environment.

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.

Awesome

- Great work providing necessary instructions to run the code in the `Instructions` section.
- All the cells in `Continuous_Control.ipynb` file should be executed to train 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.

Awesome

- Report for the project with all the details of the implementation has been provided in the submission.

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

Awesome

Great work providing the details of the implemented agent. Details of the learning algorithm used, hyper-parameters, and architectural information of the deep learning model have been provided.

- Good decision to choose DDPG algorithm for the continuous action space problem.
- Good work including model architecture in the report.
- Good work using two hidden layers in the actor and critic networks.
- Good decision to use `relu` and `leaky_relu` activations in the actor and critic networks, respectively.
- Hyperparameters you have used seem to be good.

Suggestions

You should definitely try using `batch normalization`.

To experiment more with the architecture and hyperparameters, you can check the following resources:

- Deep Deterministic Policy Gradients in TensorFlow
- Continuous control with Deep Reinforcement Learning

A plot of rewards per episode is included to illustrate that either:

- ***[version 1]*** **the agent receives an average reward (over 100 episodes) of at least +30, or**
- ***[version 2]*** **the agent is able to receive an average reward (over 100 episodes, and over all 20 agents) of at least +30.**

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

Awesome

- Discussion for the rewards is provided in the report.
- The rewards plot seems to be good and average score of +36.25 is achieved in from 37 to 136 episodes.

Reinforcement learning algorithms are really hard to make work.
But it is substantial to put efforts in reinforcement learning as it is close to Artificial General Intelligence.
This article is a must read: Deep Reinforcement Learning Doesn't Work Yet.

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

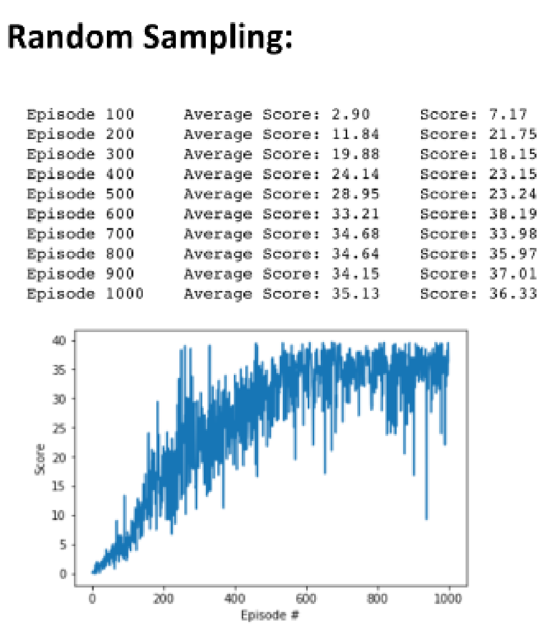
Awesome

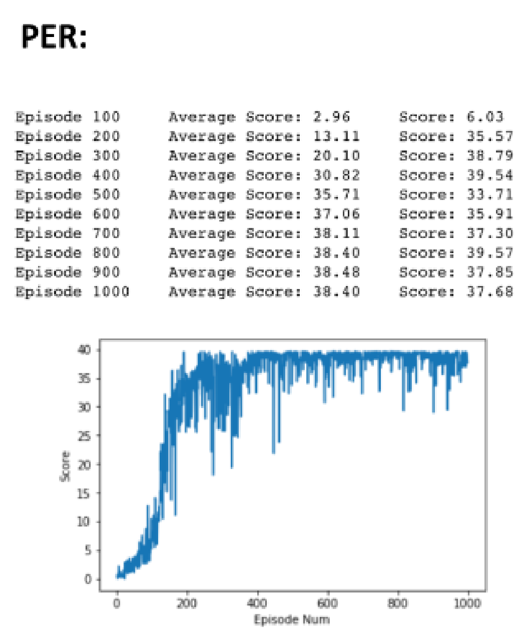
Thanks for providing the ideas for improvement.

Suggestions

An effective way to improve the performance of DDPG is by using Prioritized Experience Replay. You should check this github repo for a fast implementation of Prioritized Experience Replay using a special data structure Sum Tree.

Below is a comparison of DDPG with random sampling vs DDPG with PER for the Reacher environment. It's quite evident how episode variation decreased and performance improved.





Please check the following resources also:

- Prioritized Experience Replay
- Distributed Prioritized Experience Replay

- Reinforcement Learning with Prediction-Based Rewards
- Proximal Policy Optimization
- OpenAI Five
- Curiosity-driven Exploration by Self-supervised Prediction

DOWNLOAD PROJECT

RETURN TO PATH