UDACITY

Logout

Return to Classroom

DISCUSS ON STUDENT HUB

Continuous Control

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Dear Udacian,

Great job getting acquainted with the Deep Deterministic Policy Gradients algorithm and successfully implementing it to solve the Reacher environment. The implementation is pretty good and the environment is solve
The architectures used for the actor and critic network are decent in size with two hidden layers eac
using `relu` activation and `batch normalization`. The report is extremely informative and covers
aspects of the implementation. 😄

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

Rate this review

START

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.

Rate this review

START

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.
- The `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.

Rate this review

START

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 README file.
- The `Introduction` section describes the state space, action space, and the reward function.
- The `Distributed Training` section explains about the different versions of the environments.
- The `Solving the Environment` section explains about when the environments are considered solved.

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 and install the dependencies.

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`

Rate this review

START

section.

- All the cells in `Reacher.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 decision choosing to use batch normalization.
- Hyperparameters you have used seem to be good.

Rate this review

START

```
n_episodes=1000         # number of episodes to train
max_t=1000              # max steps per trajectory
score_target=31         # if the avg score of the last 100 episodes is above this threshold, term
inate
print_every=10         # log episode and avg. score every x episodes

# DDPG agent
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-4         # learning rate of the actor
LR_CRITIC = 1e-4        # learning rate of the critic
WEIGHT_DECAY = 1e-4     # L2 weight decay

# Models
# Actor
fc1_units=128           # number of neutrons in feed forward layer
fc2_units=128           # number of neutrons in feed forward layer
# Critic
fc1_units=128           # number of neutrons in feed forward layer
fc2_units=128           # number of neutrons in feed forward layer
```

Suggestions

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

Rate this review

START

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

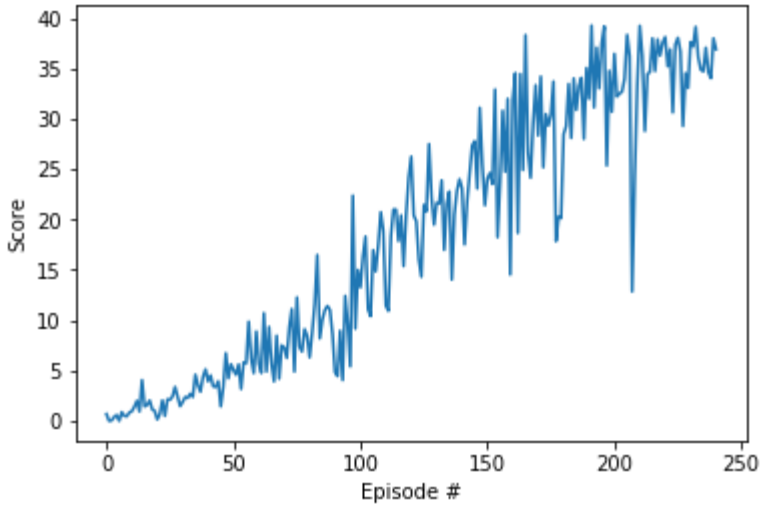
or

- *[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 +30 is achieved.



Minor correction

- The report states - "The convergence plot shows that the agent achieves a target s 30 consistently after around 170 episodes."
- Looking at the `Reacher.ipynb` notebook, it would be correct to say that the environment is solved in 240 episodes as that is when the average score over last episodes goes of +30.

Rate this review

START

- Please make this minor correction in the report.

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 following concrete ideas for improvement:
  - Solving the twenty-arms environment

Suggestions

- 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

Rate this review

START

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START