UDACITY

< Return to Classroom

Continuous Control

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Dear student, great job implementing the Deep Deterministic Policy Gradients algorithm to solve the reacher environment. Good work implementing the actor and critic networks with two hidden layers each and using `relu` activation and `batch normalization`. Performance of the agent is very good and the environment is solved in less than 352 episodes. The agent's maximum average score is higher than +38, impressive! The project meets all specifications.😄

The recent application of deep reinforcement learning to play the game *Dota 2* is a substantial step.

Also, to know more about reinforcement learning algorithms and applying these to real world problems, please do check
Deep Reinforcement Learning which is a part of the course *MIT 6.S094: Deep Learning for Self-Driving Cars (2018 version)*.

Happy learning ✌🏾

Training Code

The repository includes functional, well-documented, and organized code for training the agent.
All the necessary files and the `Continuous_Control.ipynb` notebook are included in the submission. Great job implementing the Deep Deterministic Policy Gradients algorithm to solve the Reacher environment!

The code is written in PyTorch and Python 3.

Good job completing the project using Pytorch and Python3.

You should definitely check this post comparing different deep learning frameworks: *Deep Learning Frameworks Comparison – Tensorflow, PyTorch, Keras, MXNet, The Microsoft Cognitive Toolkit, Caffe, Deeplearning4j, Chainer*

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

Saved model weights of the successful agent have been included in the github repository.

- `checkpoint_actor.pth`
- `checkpoint_critic.pth`

README

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

✅ A detailed README file has been provided and is present in the repository.

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

✅ Good work providing the details of the project environment in the `Introduction` section of the README.
✅ The provided details cover the state and action spaces, the reward function and when the agent is considered solved.

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

✅ Good work providing the instructions for downloading the project environment files in the Getting Started section
✅ Also, good work covering in detail the instructions for installing the dependencies in the Dependencies section.

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.

- ✅ The README contains the instructions to run the code in the repository.
- ✅ `Continuous_Control.ipynb` is the main file that should be used 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.

The project report is present in the submitted files.

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

Description of the learning algorithm, hyperparameters, and network architecture have been provided in great detail in the project report.

Model architecture used

2-1) Actor Network's architecture (mapping states to actions)

fc layer 1 with Batch Normalization and ReLU (input: 33 units (state_size), output: 400 units)

fc layer 2 with ReLU (input: 400 units, output 300 units)

fc layer 3 with Tanh (input: 300 units, output: 4 units (action_size))

2-2) Critic Network's architecture (mapping (state, action) pairs to Q-values)

fcs layer 1 with Batch Normalization and ReLU (input: 33 units (state_size), output: 400 units)

fc layer 2 with Concatenate[(output from layer 1, action), axis=1] & ReLU

(input: 404 units(400 + action_size), output 300 units)

fc layer 3 (input: 300 units, output: 1 unit)

Hyperparameters used

3) Parameters used in DDPG algorithm

Maximum episodes: 10000

Maximum steps per episode: 1000

First target average score: 30.0

4) Parameters used in DDPG Agent

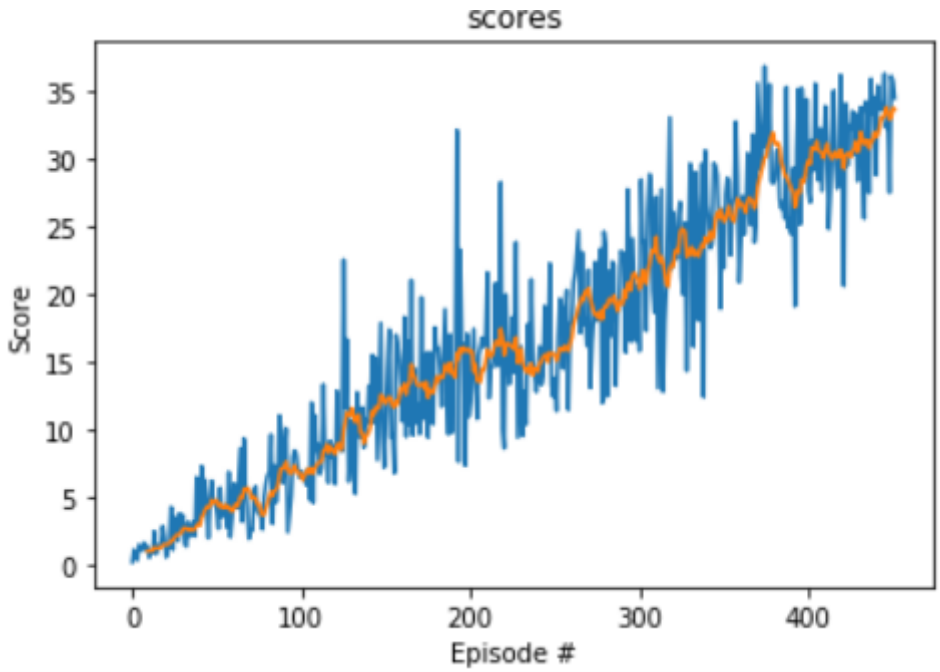
```
BUFFER_SIZE = 1e+6    # replay buffer size
BATCH_SIZE = 128          # minibatch size
GAMMA = 0.99               # discount factor
TAU = 1e-3                 # for soft update of target parameters
LR_ACTOR = 2e-4            # learning rate of the actor
LR_CRITIC = 2e-4           # learning rate of the critic
WEIGHT_DECAY = 0           # L2 weight decay
OU_MU = 0.0                # Ornstein-Uhlenbeck noise parameter
OU_SIGMA = 0.2             # Ornstein-Uhlenbeck noise parameter
OU_THETA = 0.15            # Ornstein-Uhlenbeck noise parameter
EPSILON_DECAY = 1e-6       # Decay value for epsilon (epsilon -> epsilon - EPSILON_DECAY)
```

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

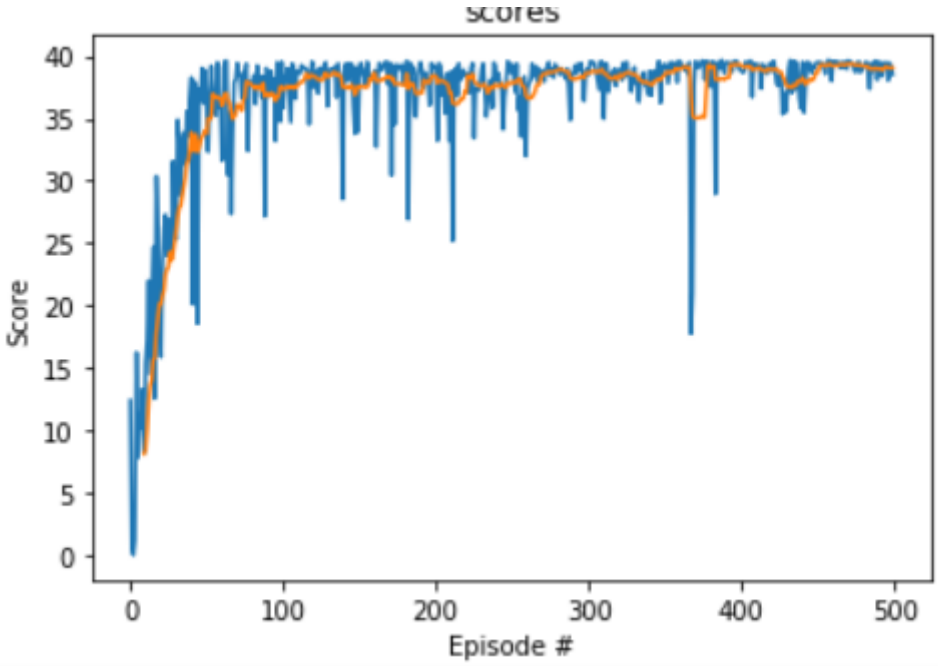
- *[version 1]* the agent receives an average reward (over 100 episodes) of at least +30, or
- *[version 2]* the agent is able to receive an average reward (over 100 episodes, and over all 20 agents) of at least +30.

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

✅ Rewards plot has been provided for the implemented agent and seems to be great.
✅ The agent solves the environment by achieving an average score of +30.05 in just 352 episodes.



☑ The agent further goes on to achieving the best average score of +38.53.



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

Idea to implement the following to further enhance the performance of the agent looks good:

- Solving the crawler environment
- Algorithms like PPO, TRPO, D4PG, A2C, A3C, and Q-Prop
- Prioritized Experience Replay
- Optimizing the hyperparameters

Suggestions

For implementing `Prioritized Experience Replay`, I would suggest you to use the `Sum Tree data structure` as it helps in a faster implementation of Prioritized Experience Replay. Please check *this implementation*.

Also, here is a useful post resource: *A Novel DDPG Method with Prioritized Experience Replay*

Also, I would like to suggest you to check the following resources as well.

- *Proximal Policy Optimization by Open AI*
- *Conquering OpenAI Retro Contest 2: Demystifying Rainbow Baseline*
- *This talk by David Silver, the author of the DQN and DDPG papers*

DOWNLOAD PROJECT

RETURN TO PATH