UDACITY

< Return to Classroom

Continuous Control

REVISÃO

REVISÃO DE CÓDIGO

HISTORY

Atendeu às Especificações

Dear student, great job. The performance of the agent is very good and the project meets the specifications. 😄

The recent application of deep reinforcement learning to play the game Dota 2 (https://openai.com/blog/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 (https://www.youtube.com/watch?v=MQ6pP65o7OM) which is a part of the course MIT 6.S094: Deep Learning for Self-Driving Cars (2018 version).

Some of the resources to read

- for continuous control using deep reinforcement learning (https://arxiv.org/abs/1509.02971)
- Towards Generalization and Simplicity in Continuous Control (https://proceedings.neurips.cc/paper/2017/file/9ddb9dd5d8aee9a76bf217a2a3c54833-Paper.pdf)
- Robust Reinforcement Learning for Continuous Control with Model Misspecification (https://deepmind.com/research/publications/2019/Robust-Reinforcement-Learning-for-Continuous-Control-with-Model-Misspecification)

Happy learning and all the very best

Training Code

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

Awesome work implementing a reinforcement learning agent to solve the "reacher" environment.

Deep Deterministic Policy Gradients algorithm, a very effective reinforcement learning algorithm. Some quick facts

- DDPG is an off-policy algorithm.
- DDPG can only be used for environments with continuous action spaces.
- DDPG can be thought of as being deep Q-learning for continuous action spaces.

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 (https://www.netguru.com/blog/deep-learning-frameworks-comparison)

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

Great work here

README

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

Great work here. Very well done. A detailed README file has been provided and is present in the repository.

Take a look at the following links to improve how your README looks

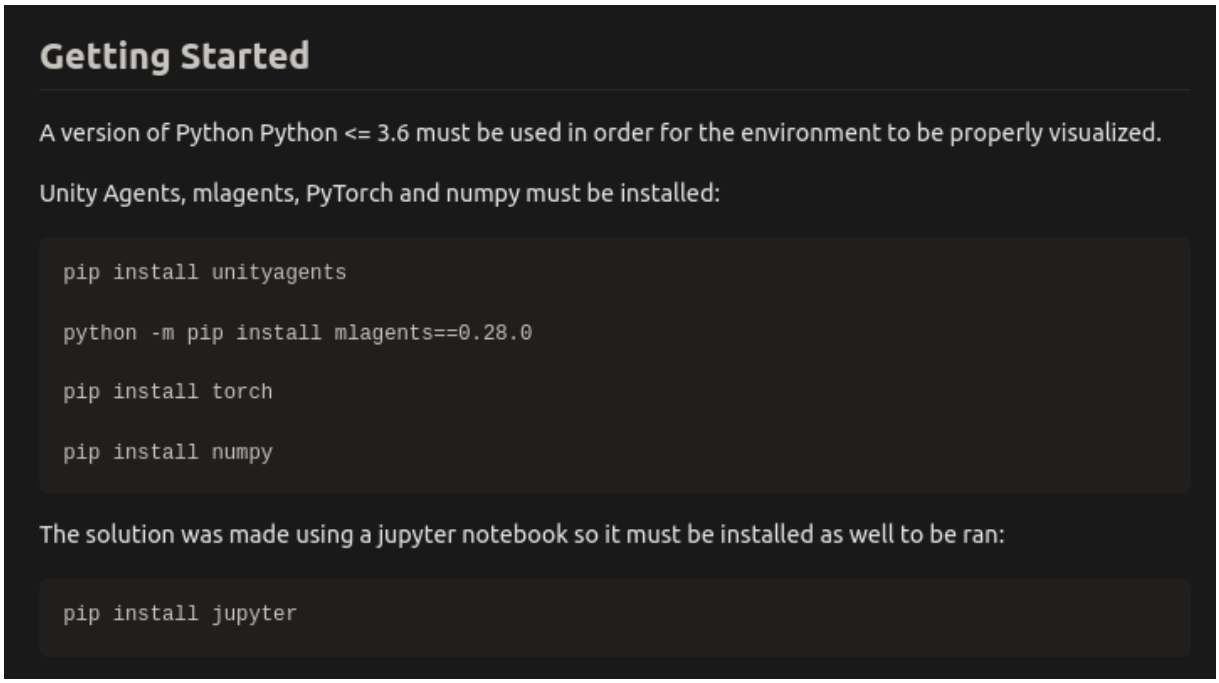
- https://blog.bitsrc.io/how-to-write-beautiful-and-meaningful-readme-md-for-your-next-project-897045e3f991
- https://dev.to/scottydocs/how-to-write-a-kickass-readme-5af9

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

Awesome work providing the project environment details including the state and action spaces, the reward function and when the agent is considered solved. The description is provided in the Introduction and Solving the Environment sections and is very informative.

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

Great work. The README talks about on how to install the dependencies and how to run the code. All the requirements are met. Very nicely done.



Getting Started

A version of Python Python <= 3.6 must be used in order for the environment to be properly visualized.

Unity Agents, mlagents, PyTorch and numpy must be installed:

```
pip install unityagents

python -m pip install mlagents==0.28.0

pip install torch

pip install numpy
```

The solution was made using a jupyter notebook so it must be installed as well to be ran:

```
pip install jupyter
```

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.

Great work. You have explained how to run the code and train the agent. Very well done. Keep up the good work

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.

Brilliant work. Report has been included in the root of the github repository.

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.

Hyperparameters

The agent receives a set of hyperparameters in its constructor which can be fine tuned in order to improve its performa

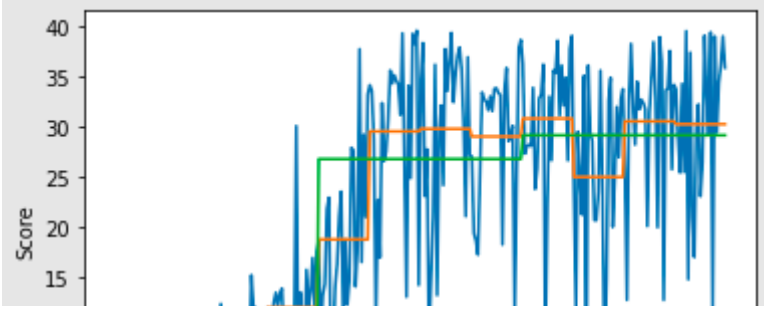
```
=====
Agent
=====
gamma (int) : discount factor
tau (int) : for soft update of target parameter
lr_actor (int) : learning rate of the actor
lr_critic (int) : learning rate of the critic
weight_decay (int) : L2 weight decay

============
Actor Critic
============
actor_fc1_units (int): Number of nodes in the Actor first hidden layer
actor_fc2_units (int): Number of nodes in the Actor second hidden layer
critic_fcs1_units (int): Number of nodes in the Critic first hidden layer
critic_fc2_units (int): Number of nodes in the Critic second hidden layer
batch_norm (bool): True to apply batch normalization
dropout_prob (float) : Dropout Regularization Probability on the Actor network - if 0 equals no dropout

=============
Replay Buffer
=============
buffer_size (int) : replay buffer size
batch_size (int) : minibatch size

========================
Ornstein-Uhlenbeck Noise
========================
add_ounoise (int) : Which episode to start using OU noise, if 0 starts from the beggining, if -1 never u
mu (float) : Ornstein-Uhlenbeck noise parameter
theta (float) : Ornstein-Uhlenbeck noise parameter
sigma (float) : Ornstein-Uhlenbeck noise parameter
```

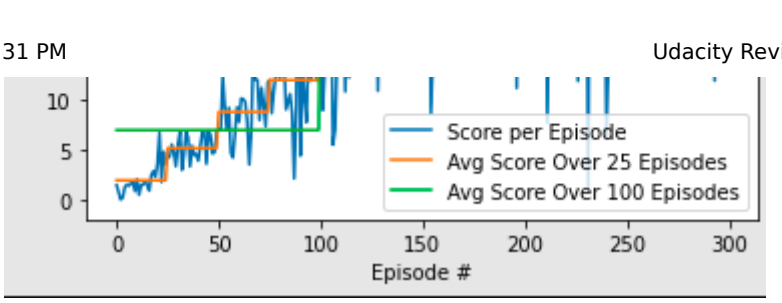
A plot of rewards per episode is included to illustrate that either:

- *[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.

Great work. Rewards plot has been provided for the implemented agent and seems to be great.





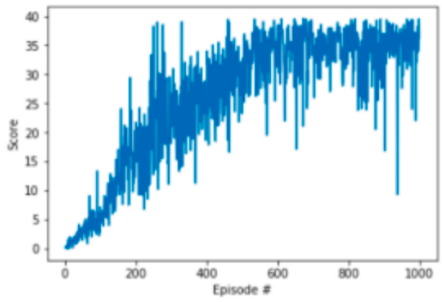
The submission has concrete future ideas for improving the agent's performance.

Great intuition. Take a look at the following as well

An effective way to improve the performance of DDPG is by using Prioritized Experience Replay which has already been implemented by you. Great work
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

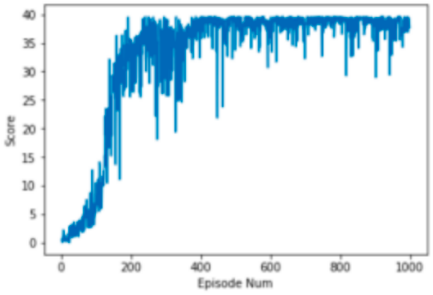
Random Sampling:

```
Episode 100     Average Score: 2.90     Score: 7.17
Episode 200     Average Score: 11.84    Score: 21.75
Episode 300     Average Score: 19.88    Score: 18.15
Episode 400     Average Score: 24.14    Score: 23.15
Episode 500     Average Score: 28.95    Score: 23.24
Episode 600     Average Score: 33.21    Score: 38.19
Episode 700     Average Score: 34.68    Score: 33.98
Episode 800     Average Score: 34.64    Score: 35.97
Episode 900     Average Score: 34.15    Score: 37.01
Episode 1000    Average Score: 35.13    Score: 36.33
```



PER:

```
Episode 100     Average Score: 2.96     Score: 6.03
Episode 200     Average Score: 13.11    Score: 35.57
Episode 300     Average Score: 20.10    Score: 38.79
Episode 400     Average Score: 30.82    Score: 39.54
Episode 500     Average Score: 35.71    Score: 33.71
Episode 600     Average Score: 37.06    Score: 35.91
Episode 700     Average Score: 38.11    Score: 37.30
Episode 800     Average Score: 38.40    Score: 39.57
Episode 900     Average Score: 38.48    Score: 37.85
Episode 1000    Average Score: 38.40    Score: 37.68
```



The following posts give an insight into some other reinforcement learning algorithms that can be used to solve the environment.

- Proximal Policy Optimization by Open AI (https://openai.com/blog/openai-baselines-ppo/)
- Introduction to Various Reinforcement Learning Algorithms. Part II (TRPO, PPO) (https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-part-ii-trpo-ppo-87f2c5919bb9)

BAIXAR PROJETO

RETORNAR

Avalie esta revisão

COMEÇAR