Collaboration and Competition

REVIEW | CODE REVIEW | HISTORY

Meets Specifications

Congratulations

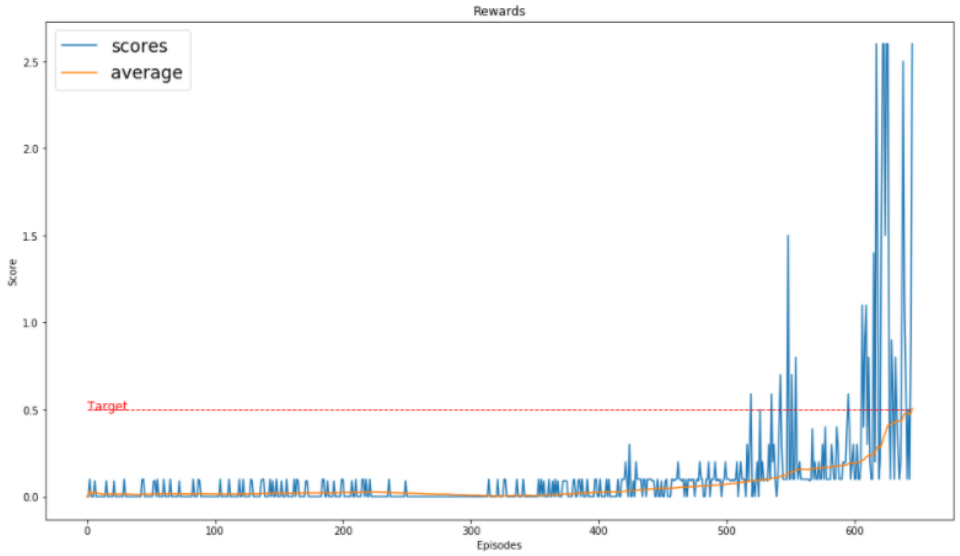
Great submission!

You have implemented and trained the agent successfully.

```
scores, averages = train(max_episodes=1500, random_episodes=250, add_noise=True) # 1500, 250
plot_score(scores, averages, TARGET_SCORE)
```

```
Training
max_episodes:1500 | max_steps:2500 | random_episodes:250
A = Average | B = Best
Episode A-Steps Mean    Moving  B-Score B-Steps Epsilon Learn   Mem %   Duration
50      17      0.013   0.013   0.100   41      0.0000  820     3.54%   0.27
100     17      0.017   0.015   0.100   32      0.0000  1714    7.11%   0.28
150     18      0.023   0.020   0.100   65      0.0000  2663    10.91%  0.30
200     17      0.023   0.023   0.100   53      0.0000  3583    14.59%  0.29
250     18      0.018   0.020   0.100   55      0.0000  4517    18.32%  0.30
300     13      0.000   0.009   0.000   15      0.9810  5229    21.17%  0.24
350     15      0.012   0.006   0.100   31      0.4319  6049    24.45%  0.28
400     19      0.036   0.024   0.100   30      0.1558  7068    28.53%  0.34
450     27      0.059   0.048   0.300   113     0.0379  8480    34.18%  0.47
500     34      0.084   0.072   0.200   91      0.0065  10242   41.22%  0.59
550     73      0.190   0.137   1.500   585     0.0002  13956   56.08%  1.26
600     74      0.199   0.195   0.800   327     0.0001  17698   71.05%  1.25
646     309     0.796   0.506   2.600   1000    0.0001  32840   100.00% 5.18
Solved

646     309     0.796   0.506   2.600   1000    0.0001  32840   100.00% 11.04
```



All functions were implemented correctly and the algorithm seems to work quite well.

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
- Introduction to Various Reinforcement Learning Algorithms. Part II (TRPO, PPO)

Training Code

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

Your code was functional, well-documented, and organized for training the agent.

Suggested reading:

- Google Python Style Guide
- Python Best Practices
- Clean Code Summary

The code is written in PyTorch and Python 3.

The code was written in PyTorch and Python 3.

Suggested reading about the discussion of what machine learning framework should be used.

- TensorFlow vs. Pytorch.
- Tensorflow or PyTorch : The force is strong with which one?
- What is the best programming language for Machine Learning?

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

You included the saved model weights of the successful agent.

Suggested reading:

- Saving and Loading Models
- Best way to save a trained model in PyTorch?

README

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

The submission included a README.md file.

I recommend you to check this awesome template to make good README.md.

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

Good work with environment details.

The README described all the project environment details.

Suggested description of the project environment details

- **Environment**: How is it like?
- **Agent** and its **actions**: When is it considered resolved?; What are the possible actions the agent can take?
- **State space**: Is it continuous or discrete?
- **Reward Function**: How is the agent rewarded?
- **Task**: What is its task?; Is the task episodic or not?

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

The README must describe all instructions for

- Installing dependencies
- Downloading needed files in Getting Started instructions item.

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.

Nice job!

I recommend you to check the Mastering Markdown, there are great tips about how to use Markdown

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.

All required files were included

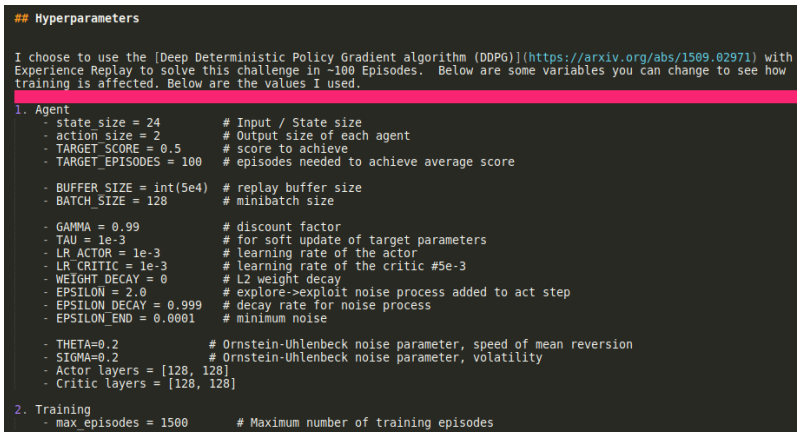
You included the report of the project with the description of the implementation.

Well done!

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

Nice job!

The report described the learning algorithm, the chosen hyperparameters and the model architectures.



```
## Hyperparameters

I choose to use the [Deep Deterministic Policy Gradient algorithm (DDPG)](https://arxiv.org/abs/1509.02971) with
Experience Replay to solve this challenge in ~100 Episodes.  Below are some variables you can change to see how
training is affected. Below are the values I used.

1. Agent
    - state_size = 24         # Input / State size
    - action_size = 2         # Output size of each agent
    - TARGET_SCORE = 0.5      # score to achieve
    - TARGET_EPISODES = 100   # episodes needed to achieve average score

    - BUFFER_SIZE = int(5e4)  # 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-3         # learning rate of the actor
    - LR_CRITIC = 1e-3        # learning rate of the critic #5e-3
    - WEIGHT_DECAY = 0        # L2 weight decay
    - EPSILON = 2.0           # explore->exploit noise process added to act step
    - EPSILON_DECAY = 0.999   # decay rate for noise process
    - EPSILON_END = 0.0001    # minimum noise

    - THETA=0.2             # Ornstein-Uhlenbeck noise parameter, speed of mean reversion
    - SIGMA=0.2             # Ornstein-Uhlenbeck noise parameter, volatility
    - Actor layers = [128, 128]
    - Critic layers = [128, 128]

2. Training
    - max_episodes = 1500       # Maximum number of training episodes
```

A report content guide:

- Description of the learning algorithm
- The hyperparameters used.
- The model architecture.
- A plot showing the increase in average reward as the agent learns.
- The weakness found in the algorithm and how to improve it.

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.

Nice work!

The report informed the number of episodes needed to solve the environment.

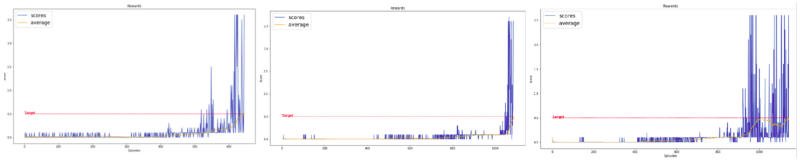
```
## Results
In my first attempt, I started with 250 episodes of random actions to then went on with the normal training.
Solved in 646 episodes.

After that training, I increased the Epsilon to 5 and reduced the decay to 0.9995 and removed the random
episodes. Solved in 1087 episodes.

The last time I trained I increased the Epsilon to 10 and reduced the decay to 0.999 and lowered the memory
size and increased the batch size. Solved in 1143 episodes.

I learned alot from this project. Most importantly take notes of hyperparameters tweaks and save results. I
also documented a lot more information to make sure training was going well, and see how much memory I was
utilizing. Also to make sure I didn't make a mistake with the scorekeeping I also monitored average steps (
A-Steps) and best steps / score (B-Steps / B-score)
```

The report informed also included a plot of rewards per episode.



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

Nice work suggesting the ideas for future improvement.

```
## Future
 ************************
1. I could had added batch normalization to keep both model's.
2. Make a function to test many hyperparameters of a smaller range of episodes / steps to get an idea of what
works best.
3. Prioritize memory replay based on reward
4. Change the hyperparameters of each agent and track them individually to see which one learns faster and
better.
5. Add the ability to train multiple passes with every memory sample.
6. Try some other algorithms.
 - [Proximal Policy Optimization (PPO)](https://arxiv.org/pdf/1707.06347.pdf)
 - [Asynchronous Advantage Actor-Critic (A3C)](https://arxiv.org/pdf/1602.01783.pdf), and - [Distributed
 Distributional Deterministic Policy Gradients (D4PG)](https://openreview.net/pdf?id=SyZipzbCb)
```

DOWNLOAD PROJECT

RETURN TO PATH